COMP 90042: Project 1

William Webber

April 13, 2014

Version history:

- *v 1.0 (30/03/2014) – Original version*
- *v 1.1 (04/04/2014)*
  - *Add two-word query to third question, to clarify that you can't assume that queries are single-word only.*
  - *Note that the Wikipedia page for "Apache Shale" is missing*
  - *Correct similarity values in example in Question 2*
- *v 1.2 (04/04/2014)*
  - *Correcting similarity score in Question 3*

1 Introduction

The goal of this project is to build a reference lookup system for online forums. A user reading a forum may select words or phrases, or even whole posts, and we will suggest relevant readings in a reference guide or encyclopædia. Eventually, we would want to be able to automatically create links from terms or phrases in the post to the encyclopædia, though we will only make initial steps towards that here.

For this project, forum posts and encyclopædia entries will be represented simply as bags of words. (Of course, more could be done with richer metadata, such as encyclopædia article titles, post subjects and threads, links between encyclopædia articles, and so forth). The encyclopædia is represented simply as a set of articles, with no structure or links between them. There is, however, a hierarchy in forums that might be exploited for more evidence. A post belongs to a forum, and a forum belongs to a community (that is, a group of forums).

The key point in the search system we are developing is that when a user launches a query onto the encyclopædia, they do so with a forum post in front of them. The query therefore needs to be interpreted in the context of the post, of the forum that the post is part of, and of the community the forum belongs to. So, for instance, if the query (say, the word the user clicked on) was "java", then if the forum was a discussion board for Indonesian politics, we would probably want to return the encyclopædia article about

the island; if the forum was for coffee drinkers, we'd want to return the article about the style of coffee; whereas if it were for programmers, the article about the programming language should be returned. We'll build up to this full functionality in stages.

2 Data set

For development, we will use (a subset of) Wikipedia as the encyclopaedia, and (a subset of) the mailing lists from the Apache Software Foundation as our forums. The system we are developing, however, should be a fully general one, into which any forums and any encyclopædia could be dropped. Don't code in solutions specific to these datasets (such as manually adding "apache" to each query).

2.1 Wikipedia dataset

I have selected 27,118 articles from the March 2014 dump of the English version of Wikipedia. This selection may be downloaded from:

`http://codalism.com/comp90042/proj/proj1/wiki.txt.bz2`

(24MB in size). The articles have had formatting and punctuation stripped, and have then been tokenized. The tokens have been case-folded, and very light stopping applied.[1] The terms are unstemmed. The processed collection is stored in a single plain-text file, one line per document. The first token on the line is the name of the Wikipedia article as in id (with spaces replaced by underscore); the remaining tokens are the terms occurring in the article.

NOTE: the name of the Wikipedia article is *not* to be used as a search field for any of these questions. (Yes, you'd get a big advantage from doing so, but the goal of these questions is to see what you can do without such metadata.)

> **Technical details:** Formatting was stripped off using the `corpora.wikicorpus` class from gensim. Tokenization was performed using the `StandardAnalyzer` class from Lucene.

2.2 Apache Software Foundation dataset

The forum dataset is taken from the full ASF mail archives up to July 18th, 2011 (available from the Amazon Web Services public dataset collection). I have selected the user-oriented mailing lists for eleven smaller Apache projects; these are listed in Table 1. This selection may be downloaded from:

`http://codalism.com/comp90042/proj/proj1/asf.tar.bz2`

(6MB in size). Forum posts were processed by stripping off all headers, then tokenizing the bodies in the same was as the Wikipedia articles.

[1] using the default Lucene stopword list; see `http://stackoverflow.com/questions/17527741/what-is-the-default-list-of-stopwords-used-in-lucenes-stopfilter`

Project	Forum	Posts	Wikipedia page	Project description
abdera	user	1017	Apache Abdera	Atom blog syndication
aries	user	573	Apache Aries	OSGi blueprint container
beehive	user	1885	Apache Beehive	Java EE framework
click	user	2073	Apache Click	Web application framework
forrest	user	5031	Apache Forrest	Web publishing framework
james	general	900	Apache James	Mail server
shale	user	2599	Apache Shale[2]	JSF webapp framework
sling	users	1483	Apache Sling	JCR content framework
synapse	user	2884	Apache Synapse	Enterprise service bus
tiles	users	1758	—[3]	HTML templates
tuscany	user	6832	Apache Tuscany	Service-oriented architecture
Total		27035		

Table 1: Wikipedia collection.

The resulting text collection is held one file per forum, a document per line, with the first token being the id of the article, and the remaining tokens being the terms from the article. (The original format of the forums, in MBOX format, is available from me on request.)

3 Questions

3.1 Question 1: Encyclopædia index

Implement, in Python, a tool which creates an inverted index of the Wikipedia articles. Term document weights are to be unit-length normalized TF*IDF scores, with the following formulae:

$$\text{tf}_{d,t} = \log(1 + f_{d,t}) \quad (1)$$

$$\text{idf}_t = \log\left(\frac{N}{f_t}\right) \quad (2)$$

$$(3)$$

where $f_{d,t}$ is the number of times term t appears in document d; f_t is the number of documents term t appears in throughout the collection; and N is the number of documents in the collection.

> **Implementation note**: The resulting index should be stored on disk, but the format you store it in is up to you. Two possibilites, both of which are fine, are using the Python `shelve` class, or simply `pickle`'ing the full index and dump to disk (use the `cPickle` module if you do this). You may *not* use any packages outside the standard Python library to implement this section. Exception: if the `shelve` package does not work on your system, you can use a replacement package.

```
$ echo -e "apache\napache aries\napache apache aries" | python qry.py wiki.db 5

>> apache
Apache_Isis 0.232290
Southern_Apache_Museum 0.216969
Apache_Excalibur 0.206819
Apache_Rocks_the_Bottom! 0.199597
Apache_Incubator 0.195659

>> apache aries
Aries 0.278531
Apache_Aries 0.250711
Apache_Isis 0.232290
Southern_Apache_Museum 0.216969
Apache_Excalibur 0.206819

>> apache apache aries
Apache_Isis 0.368171
Southern_Apache_Museum 0.343888
Apache_Excalibur 0.327800
Apache_Rocks_the_Bottom! 0.316355
Apache_Aries 0.313229
```

Figure 1: Example search output.

3.2 Question 2: Querying the encyclopædia

Implement a query interface to the index built in Question 1. For the query vector, the weight $w_{q,t}$ of query term t should be calculated as:

$$w_{q,t} = \log(1 + f_{q,t}) \tag{4}$$

where $f_{q,t}$ is the frequency of the term in the query.

The implementation should take the index and the number of results to return per query as command line arguments, then read a query a line at a time from standard input, and print the results to stdout, as "article-name score". An example invocation and output from this interface is given in Figure 1. (It is simpler to place your queries in a file, then `cat` them to program; I've used `echo` here just for illustration.) Place your results for the same queries, but to depth 10, in your answer.

3.3 Question 3: Pivoted length normalization

Note that the first result in Figure 1 for the query "Apache" is "Apache_Isis", rather than the "Apache" main page. Why is this? Similarly, when we include "apache" twice in the third question (trying to bring "Apache_Aries" above "Aries" in the answer), we again get "Apache_Isis" on top.

To try to fix this, implement the pivoted document length normalization proposal of Singhal, Buckley, and Mitra[4], as described in Section 2.3 of that paper. Specifically, under unit length normalization, the normalized weight of term t in document d is:

$$\widehat{w}_{d,t} = \frac{\text{tf*idf}_{d,t}}{|\vec{V}_d|} \tag{5}$$

where $|\vec{V}_d|$ is the length of the (TF*IDF) document vector for document d. Under pivoted length normalization, the weight changes to:

$$\widetilde{w}_{d,t;s} = \frac{\text{tf*idf}_{d,t}}{(1-s)\left(\sum_d |\vec{V}_d|\right)/N + s|\vec{V}_d|} \tag{6}$$

where $\left(\sum_d |\vec{V}_d|\right)/N$ is the average document length, and $0 < s < 1$ is a user-tunable parameter. What is the effect of increasing s? What happens if we set $s = 1$?

> Implementation note: there are two ways you could do this. You can either calculate the unit length normalization at index time, and store $\widetilde{w}_{d,t;s}$ in the index, with a fixed s. Or you can store tf*idf in the index, as well as document lengths in a separate lookup table, and calculate $\widetilde{w}_{d,t;s}$ at query time. Implement it whichever way you prefer. But whichever way you implement it, s should be a command-line option, not hard-coded into your implementation.

Run the queries in Figure 1 to depth 10, with the slope parameter set to $s = 0.5$, and place your results in your answer. (When I run the query "apache", I now get "Apache_(disambiguation)" as the top result, with score 0.157835.)

3.4 Question 4: requiring all query terms

If you look at the results to the query "apache aries", shown in Figure 1, you'll note that some of them have nothing to do with "aries" (the result documents do not, for instance contain the term). You'll probably observe the same behaviour for the pivot-normalized results as well. The system does this because the "apache" match is so strong that the lack of an "aries" match is overlooked. This behaviour, however, is often confusing to users. Many search systems modify the statistical matching algorithm to require that every matching document contains all of the query terms, at least for short queries. Add an option to the query evaluation system you built in Question 3 that specifies that it should run in this mode. Make this option settable on the command line. Run the queries:

- apache james
- apache forrest
- apache aries

with and without this option turned on, and show the results.

[4]Singhal, Buckley, and Mitra, "Pivoted Document Length Normalization", *SIGIR*, 1996; `http://dspace.library.cornell.edu/bitstream/1813/7217/1/95-1560.pdf`.

3.5 Question 5: designing disambiguation by source context

Now that we're reasonably happy with the encyclopædia search system in isolation (though certainly there is more that we could do to improve it), it's time to solve the larger problem. If you run the names of the ten projects in Table 1 against the Wikipedia search system (that is, "abdera", "aries", etc., except for "tiles"), you'll find that the page for the Apache project never turns up first, and only twice turns up in the top 5 (third for "abdera", and fifth for "tuscany"). Yet a user clicking on these words in a posting on an Apache mailing list would almost certainly expect the Wikipedia page for the named project to come up first, and related pages near it. Similarly, searches for more general terms that have special meanings in a computer context, such as "request", "message", "file", "service", "class", "public", or "string", frequently produce context-inappropriate results ("String quintet repertoire" for "string", for instance).

Drawing upon the content of the course up to and include Lecture 7 ("Singular value decomposition"), describe three different methods that could be used to add context to the searches, and so return more appropriate results. (Manually adding "apache" to each query is not a suitable method, as the system is meant to be general-purpose.) The three methods may be independent, or one may be a variant of another. Give consideration not just to the technologies you might apply, but to the sources of evidence you might use. You should assume that the user always has a single forum post open in front of them when they launch a query. You may not, however, assume that the query is taken from a particular bit of text within the forum post.

3.6 Question 6: implementing disambiguation

Take *one* of the disambiguation methods you've designed above, and implement it. For this question, you may use the facilities of `gensim`, `scipy`, or a similar package, if required. Show the output of a search for each of the eleven project names given in Table 1, and also for the queries "request", "message", "user", "file", "service", "class", and "file request". Where your method uses contextual evidence from a particular forum or post, please specify the forum and/or post that you use for context for each of the above queries.

4 Project details

4.1 Submission format

Project submission takes two forms: text and code.

4.1.1 Text

Prepare a written document that describes your answers to each of the above questions. Where the question requires an implementation in code, give a description of how the implementation was designed and made. The document should include answers to running questions in the project specification, as well as the output of searches, where required.

The written document should be in one of the three formats:

- Plain text
- HTML (with all required included files)
- Latex (compileable with PDFlatex)

You will not loose marks for using only plain text. Do *not* submit a Word document, or any other binary format (including PDF). If you are using Latex, do *not* submit the generated PDF or any intermediary files.

4.1.2 Code

Code is to be neatly organized, adequately commented Python code. It is preferable that you develop one package that is able to perform all of the above functions, with appropriate options to specify which is required. See the Python `optparse` and `argparse` modules for processing command-line options. Please clearly document the code that implements Question 3 (pivoted document-length normalization) and Question 4 (requiring all query terms).

4.2 Submission method

Submission is to be via Subversion. Create a directory marked "proj1" at the top level of your Subversion working directory. You may use whatever directory structure beneath that you like, as long as it is clear where the text and where the code are. Do not place source data or generated indexes into Subversion. Check in your code and text regularly while you work on it.

4.3 Deadline

The due date for this project is Sunday, 27 April, at 11:59pm. Whatever is committed in Subversion by then will be taken as your project submission.

4.4 Individual work

This project is to be completed as individual work, not as a team project. You may discuss high-level questions, but do not share your code or your text with other students.