Summary of Testing

Taylor Brown: tab23

April 30, 2019

1 Introduction

This document summarises the various tests that were performed on the system built in the project *A collection of scientists in Wales between 1804 and 1919 built using Natural Language Processing techniques*. The system is comprised of three different parts, which make up the three different sections in this report.

This system looks at the building of a collection of scientists from newspaper articles published approximately 100 years ago. Articles are downloaded and saved as local files. The article text is analysed and scientists are extracted. Scientists, articles and the different actions that scientists did, the reason they are in the newspaper, are all stored in a database.

The user interface of this system is a web application, built with flask, where a user can search for a name, title of an article or field of study. These searches query the database, and return a table filled with the results. Any search which returns an article ID returns the ID as a link to the article on the National Library of Wales newspaper archive.

1.1 Testing Scope

Covered here are the unit tests for the information extraction software, manual inspection of the information in the database, and webtesting for every aspect of the web app.

1.2 Iterations

The project was developed in a series of sprints, each one improving the system slightly in a different way. Because of this, testing was done regularly and in small batches.

1.2.1 Information Extraction

Unit tests were written for this section whenever a new method was produced that required tests. These tests were ran regularly, and used to determine that a method functioned in the way it was thought to function. They were also updated whenever a method was changed in any way - this includes slight changes to structure as well as complete rewrites. After a change was made, a method was not left alone until it passed its unit test.

1.2.2 Database

The database was manually inspected whenever more data was added to it. These inspections were made to ensure that the data populating the database was of the form and format expected, and checked that newspaper and article titles were of an appropriate length for the constraints set. Given that new data was added in batches regularly, this inspection occurred often.

1.2.3 Web App

Though no unit or system tests were written for this part of the project, the web app was tested manually for functionality - in particular, to ensure that any changes made to the application did not have an adverse affect on reading from the database and displaying the results, since that is the key role of the app.

2 Information Extraction Testing

Unit tests were written for the below list of tasks that this software was designed to complete. These tests **must** be passed before further development can occur. When changing one of these methods in any way, the test must be changed accordingly.

- Documents break down into articles
- Articles break down into text
- The spelling of words is corrected
- Words are tagged with correct part of speech
- Entities are identified
- Names are recognized as names
- People are created

The integration tests for this system look at how various parts of the software work together. The system is broken down into a collection of methods that work together to achieve a defining point in the software. These points are:

- Text collection: tests the findText(), word_tokenize() and correctSpelling() methods together
- People: tests the tag(), entities() and createPeople() methods.
- Duplicitiy: tests that mentions of the same person in the same article are removed like they should be. Tests that articles that appear in more than one field of study are removed.
- Database: tests that the database is populated with results correctly.

The difference between these tests and the unit tests are that, here, it is already assumed that each method works as expected: integration testing would not have commenced unless the methods passed unit testing. Each test is written as a black box: there is an input of some kind, the methods are run sequentially without external interferance, and the output is compared to expected values.

The text collection test uses the *correctSpelling()* method, which causes the test to fail approximately 50% of the time. This is a known issue: if a single test fails with the error *AssertionError: Lists differ: ['POR[126 chars] 'felix', 'Board', 'School', ',', 'has', 'won'[931 chars] '.'] != ['POR[126 chars] 'felon', 'Board', 'School', ',', 'has', 'won'[931 chars] '.']*, it is to be ignored.

3 Database

The database was filled with small, sample sets of data initially. These sets were extracts taken from the json files already gathered to test the information extraction, and allowed for any obvious bugs in the population code that could be spotted, fixed and reran without waiting for hours at a time. In this way, it was discovered that the article title field was too small to allow for all potential titles. This field was doubled in size twice before this particular error stopped appearing.

There is only so many issues that will be made known using small sets of data, and one such problem that remained undetected throughout the testing of the database was the occasions where the code would attempt to store an article or person twice. After some inspection of the specific articles causing these problems, the fault was determined to lie with the way articles are written: multiple mentions of the same person in the same article caused them to be saved as the same scientist, and an article appearing in multiple fields of study meant they were saved in both article collection files. This wouldn't have been a problem if the primary key for the articles table was generated, but this field used the article id number, given to the articles by the National Library archive, so this caused another bug. Because both bugs were exceedingly rare, there was a very small change that the test data would trigger them; unfortunately, they did not, and the whole program had to be run again from the start.

4 Web App Testing

Various different aspects of the web app require regular testing to ensure that they work consistantly.

- The different kinds of search that can be made
- The handling of unexpected search strings
- The return of information
- The format of returned information
- Navigation around the web app

As seen in 1, the different searches were tested in a variety of ways. Because each search is handled in the same way within the flask app, it would be fair to assume that each search within a selected field that is deemed 'incorrect' would be handled the same way: Every potential search has been tested to ensure that this is the case.

- The example name is *Robert Smith*
- The example field is *Chemistry*
- The example title is *Death of Professor Magnus*

Table 1: Testing the searches

Test ID	Section	Test	Expected Result	Outcome
T-S1:1	Name	Search for Full Name	Results displayed that exactly match the search term: *Robert Smith*	Seven results displayed for a Mr Robert Smith, his field and the articles the name appears in.
T-S1:2		Search for partial name	Results displayed that match the search term *Smi*	Sixteen results displayed for various “Smiths”’. The same results as T-S1:1 appeared as well.
T-S1:3		Search for field of study	Redirect to home page	Redirect back to home page
T-S1:4		Search for article title	Redirect to home page	Redirect back to home page
T-S2:1	Field	Search for full field	Results displayed for every entry with that field	List of people displayed with a connection to chemistry
T-S2:2		Search for incomplete field	Results displayed that match the field that should have been typed (Test uses *chem*)	Results displayed that matches the full search meant.
T-S2:3		Search for unavailable field	Redirect to home page	Redirect to home page
T-S2:4		Search for name	Redirect to home page	Redirect to home page
T-S2:5		Search for article title	Redirect to home page	Redirect to home page
T-S3:1	Article Title	Search for full article title	Results displayed for all articles with that title	Three articles returned that match the searched title
T-S3:2		Search for partial title	Results displayed for full titles that match the search term *Professor Mag*	The same three articles returned for the partial match as for T-S3:1
T-S3:3		Search for name	Display list of articles with name in the title, or redirect to home page if none match.	Redirect to home page
T-S3:4		Search for field of study	Display list of articles with the field of study in title, or redirect to home page if none match.	Displays list of articles with chemistry in title

T-S3:3 and T-S3:4 suggest that some sort of system designed to filter results may be required should the amount of data made accessable by this system grow much more: article titles can have any word combination in them, so finding something specific in a large result set could be difficult. Though all of these tests have passed, it was concluded that some method of informing the user that their search could not find any results might be useful, to make the transition between submitting the search and redirecting back to the home page less jarring. A dialogue box might be the most useful way to do this.

Another test that is run manually is activating several of the hyper links after search results are displayed, ensuring that they link back to the correct article. As of writing, this feature does not work. Why this does not work is understood.