GitHub Repository Search Application

Student Name(s)	Connel McGovern	Eoin Murphy
E-mail Addressed	connel.mcgovern22@mail.dcu.ie	eoin.murphy74@mail.dcu.ie
Student Number(s)	11379746	11487358
Programme	Computer Applications (CASE4)	Computer Applications (CASE4)
Module Name	Search Technologies	Search Technologies
Module Code	CA4009	CA4009
Submission Due Date	11/12/2015	11/12/2015

Name: Connel McGovern Date: 09/12/2015

Name: Eoin Murphy Date: 09/12/2015

Abstract

In this report we will be describing our proposed search application. In designing our proposed system we will look to enhance the current functionality of the GitHub repository search. We determined the possible users for this search application and the search functions that they will require. To do this we looked at the current system in place on GitHub and how our search application will differ. We looked at the possible additional features of our system and the overall constraints of our application.

In our functional description we describe the additional features to be implemented and how we would go about doing so. Within this section it is described how we implement our document collection, methods of conflation, our indexing process, scoring of documents for our best-match model and the ability for users to provide additional ranking for required repositories using the ranking functions implemented within our application. This ranking allows the users to have a direct influence on the the relevance of feedback.

The final section shows how the system will look when finally implemented and how each component of the system will perform and interact. Additionally we describe our plan on evaluating the system. We discuss how we will use a test collection of data to apply queries to allow us to predict the accuracy of our information retrieval within our application. Throughout the document we used external sources to determine what algorithms and methods we would use in designing the system.

1. Overview

1.1 Our Search Application

For our Search Technologies (CA4009) project we aim to design a search application for the GitHub website. GitHub is a repository hosting website designed specifically for software development. It allows users to upload and manage repositories of source code and documents. It allows users to view and provides access control features to allow for collaboration of various project repositories.

Our search application is to be based on the personalization and adaptation of information retrieval and will also make use of some features of social network search for the GitHub website. We wish to allow users to be able to search the website for relevant repositories, specific user accounts and search for projects written in a specific programming language. The aim of the application will be to enhance the functionality of the search system currently implemented on the site.

1.2 Application Users

This application will be designed for a specific group of predicted users. We predict that the users will include students, software developers, and employers. We feel that they will require a search feature to allow them to search for software application ideas, specific programming language solutions and where applicable to search for possible future employees based off GitHub user profiles.

1.3 Current System vs Proposed System

The current search system allows for a keyword search. A limitation to this keyword search would be that it only utilizes text retrieval from a repositories title and description. For our proposed system we would wish to build upon the current search system in place and to additionally add text retrieval for a repositories associated README document. This would allow for a broader range of keywords to search for a relevant repository.

Currently the search system does not allow for grammatical errors and will only search the exact word that the user inputs. For our proposed system we would wish to implement conflation which allows for comparison of similar words using various different methods, as explained within the implementation section.

1.4 Additional Features of Our Search Application

Content based document indexing will also be required to be implemented within our search application system structure. Indexing allows for improved speed and accuracy of document search and information retrieval.

The documents returned from our search application would also have a ranking function applied to them using term weighting. This ranking would allow the results to be ordered in terms of relevance in descending order. The most relevant being places at the top of the returned list.

1.5 System Constraints

A notable constraint for our proposed system may include repositories that do not have associated README document. This issue would result in a negation of our proposed solution as we would be required to only apply text retrieval to the title and short description. This may result in poor performance in the search application as in general the descriptions provided by users can be very brief. In this case a huge emphasis would be placed upon our metadata retrieval to allow for sufficient results for the search application.

<u>2. Functional Description</u>

The following section describes the different components that relate to the overall functionality of our search application and how each function is implemented within our proposed system:

2.1 Repository Document Collection

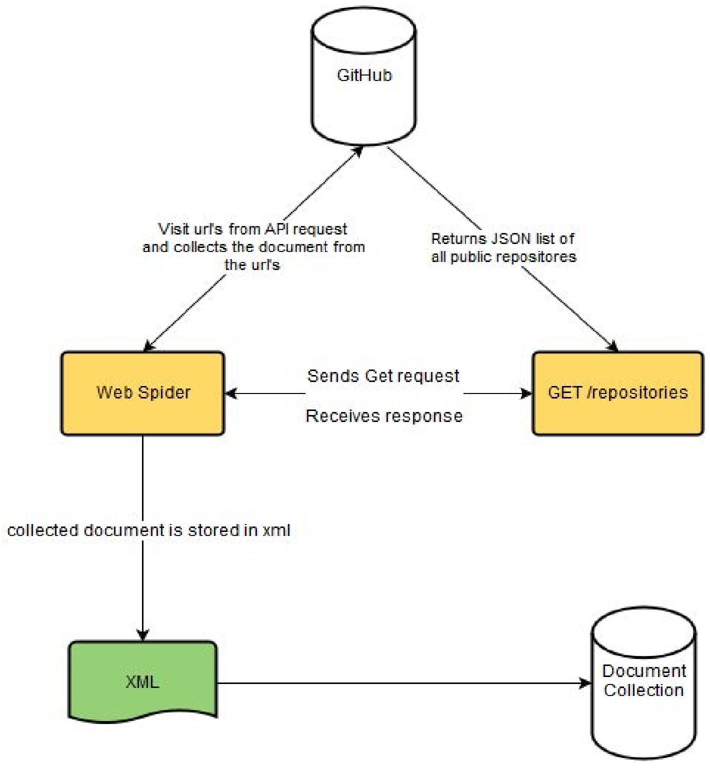
2.1.1 Functionality of Document Collection

The first function our system must be able to perform is the collection of GitHub repository documents. The repository document is collected from the repositories HTML home page on GitHub. Repository home pages include the following sections; title, description, README document, owner username, count of times star'd [2a] and a breakdown of the languages contained in the repository. To collect the repositories we employ web crawlers (spiders) to crawl the github network and to scrape all public repositories documents. These documents are then stored in an xml file format for the later specified methods of conflation to be applied and then for these files to be indexed.

[2.a] Github allows users to 'Star' a repository as a way for the community to rank repositories by highlighting that they have good content.

2.1.2 Implementation of Document Collection

To get the list of all public repositories the system makes use of GitHub's public API. Specifically, the API call 'GET /repositories' [2.1]. The call provides a JSON dump of all public repositories in the order they were created. Our web crawler then collects the documents located at the specified URL. It is extremely useful for our system that the JSON dump is ordered by creation date as it allow us to store the the latest repository in terms of creation date for each collection run. When the system goes to collect newly created repositories to add to our collection it can do so with ease. This will eliminate the collection of any unnecessary documents already in the system.


GitHub
Visit url's from API request and collects the document from the url's
Returns JSON list of all public repositores
Web Spider
Sends Get request
Receives response
GET /repositories
collected document is stored in xml
XML
Document Collection

2.2 Conflation / Stop Word Removal

2.2.1 Functionality of Conflation / Stop-Word Removal

The next function that our system must carry out is to use stop word removal and conflation on the collected repository documents. Our system uses a predefined list of stopwords [2.2] to determine which words to remove. Before we begin applying conflation to the document the system first applies case-folding to the document by reducing all letters to lowercase. The reason for this is to help in term match by not having capitals skew results (ie. allows instances of words at the start of a sentence to match with a query word which is in all lowercase). Next our system applies stemming to the words in the document. The algorithm we propose to use for this is *M.F Porter's Algorithm* [2.3] Also known as the Porter stemmer. The reason for this choice of algorithm is it has been shown to be very effective at stemming and shows very good results when compared with other algorithms [2.4]. This algorithm is based on the idea that suffixes in english are mostly made up of a combination of smaller and more simple suffixes. The porter stemmer works by defining a set rules to handle the reduction of different suffixes. It uses five phases for word reduction that are applied sequentially where each phase uses various confections to select which rules to follow.

2.2.2 Implementation of Conflation / Stop-Word Removal

For implementing conflation to our documents the system first applies a form of string-similarity matching to account for grammatical errors. The method used is approximate string-matching using the Levenshtein distance [2.5] . After this is completed words that are found in our stop word list are removed and then the porter stemming algorithm can be applied.

2.3 Indexing of Repository Documents

2.3.1 Functionality of Indexing of Repository Documents

This section describes the function of indexing the collected repository XML documents. The indexing method we use is a modification of the “inverted index” as described in “Daniel Egnor” and “Robert Lord’s” paper “structured information retrieval using XML” [2.6].
The reasons for this choosing this method of indexing is it enables us to return snippets from the document with relative ease and also it allows for us to modify the index structure to another extent that enables the system to weight certain terms higher based on their location in the document. This is expanded upon in further detail in section 2.5.

2.3.2 Implementing Indexing of Repository Documents

After we have collected the documents and conflation / stop word removal is applied we then index the documents using the modified inverted file index as follows:

```
found_in(term, path, ID)
occurs_in(ID, document, address)
```

Where the newly added variables are:

path: is the xml path to the node the term is found in

ID: is the unique ID for the term

address: is the location in the xml document the term occurs in the form [node num, offset in node]

2.4 Best-Match Model

2.4.1 Best-Match Model Functionality

For our application we have decided to use a Best-Match model. The reason we choose this over an exact-match is we want our system to return the results in a ranked order where as with exact-match it documents either just matches or fails to match the query. The best match model we use is the vector space model which allows us to score documents based on the query. For calculating term weights we use the weighting scheme lnc.ltc where the first triplet gives the document weight and the second triplet gives the query weight [2.7].

l = log-weighted term frequency : $1 + log(tf)$

n = no idf

c = cosine normalization : $\sqrt{\Sigma\, w_i^2}$

t = idf : $log(N/df)$

So from this we can see our scheme defines the document weight as log-weighted term frequency with no idf and cosine normalization and our query weight as log-weighted term frequency with idf and cosine normalization.

The equation that is fundamental to score documents in the vector space model is [2.8]:

$$score(d,q) = \frac{d \cdot q}{|d| \cdot |q|}$$

From the this equation we use our weighting scheme(lnc.ltc) to give us our equation we use to score our documents [2.9]:

$$score(d,q) = \frac{\Sigma\, d_i \cdot q_i}{\sqrt{\Sigma\, d_i^2} \cdot \sqrt{\Sigma\, q_i^2}} = \frac{\Sigma(log(tf)+1) \cdot ((log(qtf)+1)log(\frac{N}{df}))}{\sqrt{\Sigma(log(tf)+1)^2} \cdot \sqrt{\Sigma(log(qtf)+1)log(\frac{N}{df})}}$$

Where d = document, q = query, tf = term frequency, qtf = query term frequency, N = number of documents and df = document frequency.

2.4.2 Best-Match Model Implementation

Each documents term frequency vectors are calculated and stored. When a query is entered by the user our conflation methods are applied to the query and the query term frequencies are calculated. Using the above equation documents are given a score in relation to the query entered, the system then returns all documents matching in order of the score calculated from highest to lowest. The document only performs this matching on the text contents contained in the Title, Description and README xml nodes.

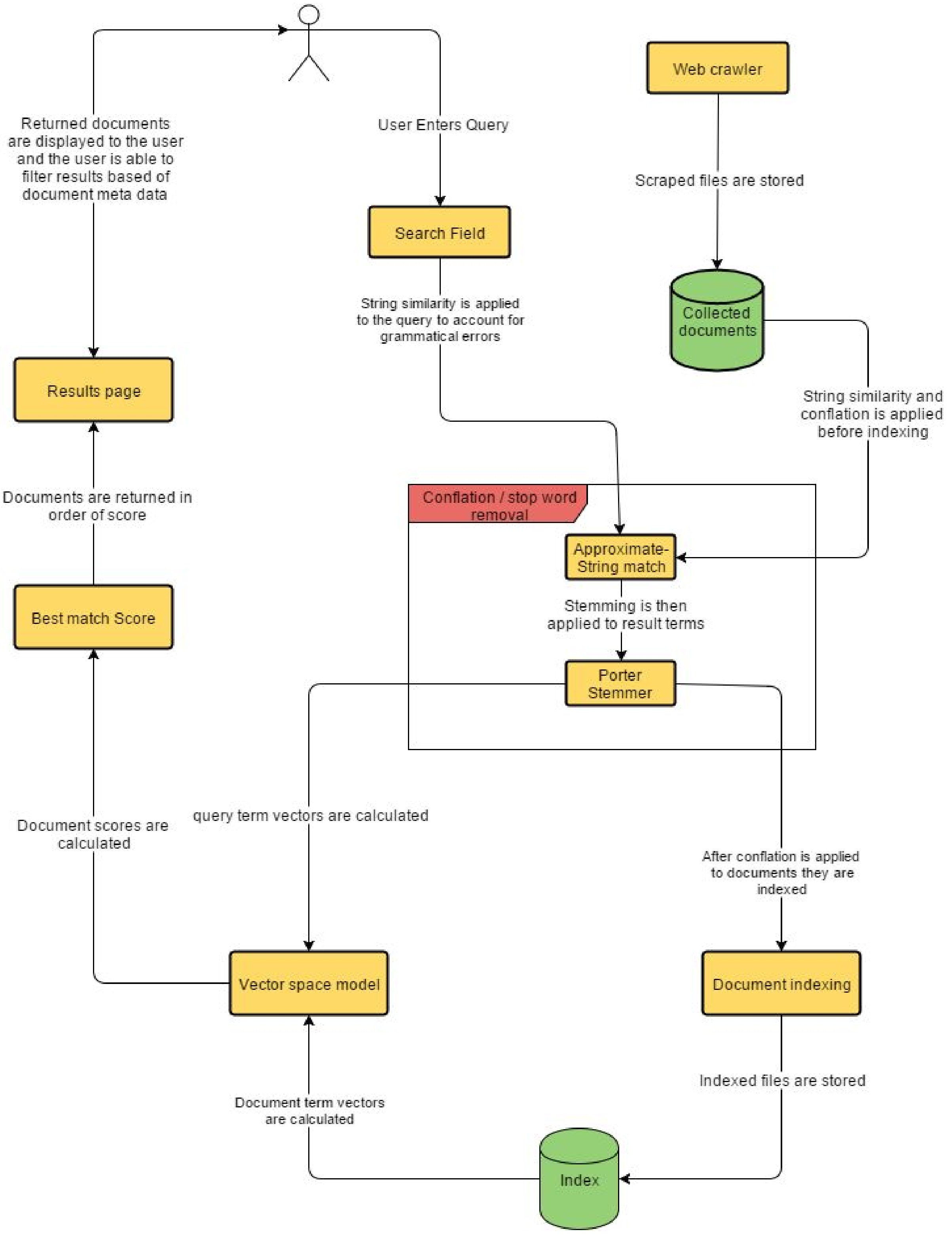
2.5 Using Metadata in Aid of Results Ranking

To aid in ranking documents we make use of the metadata contained on the repository. From the document collection the metadata is stored in specific XML locations. Our system is to provide users with the capability to filter results that only include certain matches in the metadata. What this means is users will be able to chose the value of a metadata attribute that will result in all returned repositories that do not contain the value in that attribute to be remove from the results. For example if a user selects the attribute Language is to be java all results that do not have this value for language are removed from the results.

3. Implementation and Evaluation

3.1 Implementation

The following diagram depicts the implemented system:


Returned documents are displayed to the user and the user is able to filter results based of document meta data
User Enters Query
Web crawler
Scraped files are stored
Search Field
String similarity is applied to the query to account for grammatical errors
Collected documents
Results page
String similarity and conflation is applied before indexing
Documents are returned in order of score
Conflation / stop word removal
Approximate-String match
Stemming is then applied to result terms
Best match Score
Porter Stemmer
query term vectors are calculated
Document scores are calculated
After conflation is applied to documents they are indexed
Vector space model
Document indexing
Indexed files are stored
Document term vectors are calculated
Index

3.2 Evaluation Plan

The evaluation of our proposed system is of huge importance as to validating how well our system is designed. In this section we will look to examine how effective our search application is in terms of the retrieval of the required information for a user based off a query.

For our evaluation plan we will determine effectiveness using a collection of data, specifically GitHub repositories and a number of different search queries to replicate queries similar to that of proposed users for our search application. Once a query is applied to the data collection an ordered list of responses will be returned, referred to as the response set.

In our evaluation we will firstly input the test collection of documents for different repositories into our search system. Evaluating whether or not documents within the response set are relevant a binary assessment will be applied, with the expected outputs to be either "relevant" or "not relevant".

We will apply a test case of document to be retrieved for a specific query. This will allow us to see which documents the system retrieves and will allow us to compare them to the documents that we deemed relevant prior to the system search.

The performance of the information retrieval system can be evaluated using a number of different methods, primarily precision the method. Applying the precision method will allow us to determine the percentage (given in fraction form initially) of documents retrieved that are relevant to the user's needs.

We would also supply a feedback form to a number of test users to define whether documents retrieved are deemed relevant. This will allow proposed users for the system to provide feedback as to the performance of the application.

4. References

2.1: GitHub Developer Home, Repository API, Developer.github,

2.2: Onix Text retrival toolkit, Stop words List 1,

2.3: Porter M.F, An Algorithm for suffix stripping, 1980

2.4 Anjali Ganesh Jivani, A Comparative Study of Stemming Algorithms, University of Baroda Vadodara, Gujarat, India, 2011

2.5 V. Levenshtein. Binary codes capable of correcting spurious insertions and deletions of ones. Problems of Information Transmission, 1:8 17, 1965.

2.6 Egnor Daniel, Lord Robert, Structured Information Retrieval using XML, XYZFind Corp, Washington USA

2.7: Manning Christopher D, Introduction to information retrieval 6.4.3, 2008, Cambridge university

2.8: Manning Christopher D, Introduction to information retrieval 6.3.2, 2008, Cambridge university

2.9: Callan Jamie, Retrieval models: Boolean and Vector space, 28, Carnegie Mellon University

3.2 Manning Christopher D, Introduction to information retrieval 8.4, 2008, Cambridge university

: Manning Christopher D, Introduction to information retrieval 4.2.4, 2008, Cambridge university