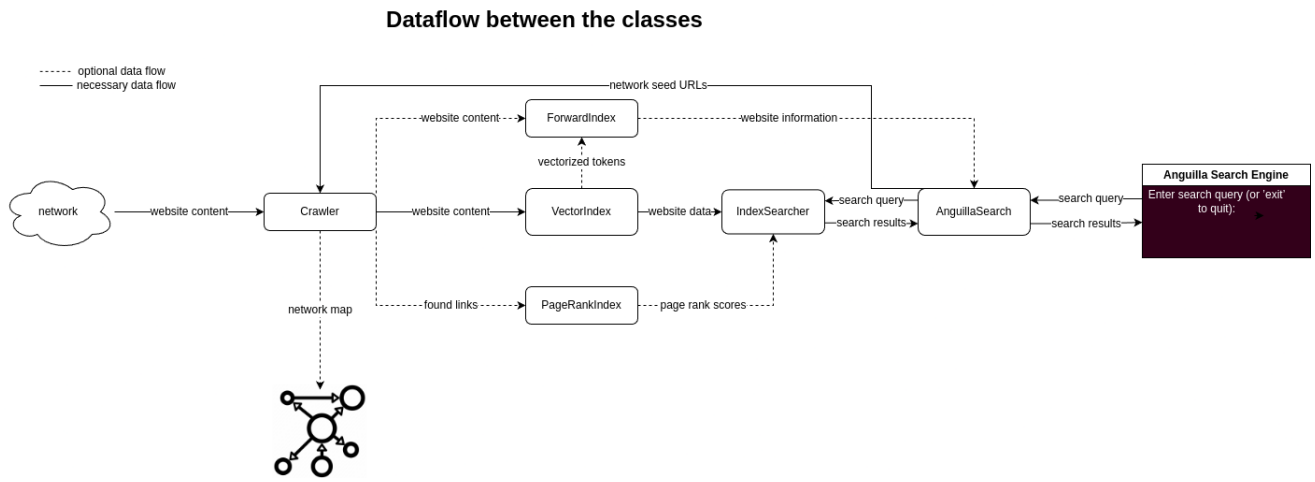
Anguilla Search PDF documentation

Nico Beyer

Overview and general information



The general data flow for this project is shown above. We will explore the classes as shown from left to right.

Crawler.java

The flow starts with the crawler class. The crawler gets a set of links(seed) and visits each linked website then separates the data of that website into links and text content. The links are stored to visit next. The bare crawler class can map out a portion of a network showing connections between the first 16 websites encountered from the seed. Links and encountered websites can be processed to create a map of the network and exported as a diagram. The crawler class may also be used to gain information about a network such as number of websites reachable from the seed and total link count of those pages. Text content of the found websites may be passed on to the ForwardIndex and/or VectorIndex links may be passed on to a PageRankIndex for further processing and enabling search operations on the data.

ForwardIndex.java

The ForwardIndex class gets a websites text content and stores it. The saved text content can later be accessed by URL. This can be used to access parts of the website data later on for example finding the title of a website that has been found in a search.

VectorIndex.java

The VectorIndex class acts as a reverse index. It gets a websites text content and stores it. Further if a ForwardIndex is provided it will add a vectorized version of each added websites content to that ForwardIndex.

PageRankIndex.java

The PageRankIndex gets a websites links and calculates the relevance of each given website by how many times it is referenced by other websites. The score quantizes how relevant other web content creators in the network think this websites content is. Each saved score can be accessed by providing the websites URL. The page rank can be used to refine search results.

IndexSearcher.java

The IndexSearcher class is a wrapper for the VectorIndex which enables search functionality on the index data. It may use a PageRankIndex for a refined search operation.

AnguillaSearch.java

The AnguillaSearch class sets up the Crawler, ForwardIndex, VectorIndex and PageRankIndex and handles input/output for the user.
When started a network will be crawled. After the indices are created the user may enter search queries and will receive the formatted search results until 'exit' is entered.

extra feature(Z3.1)

When searching the VectorIndex with an IndexSearcher a weight-table may be given to apply weights to a given search query allowing for more detailed queries.
Each word of the query may be assigned a weight value from [0.0 - 1.0] representing percentage of the importance in the entire query. 0.0 would mean ignore this word. The sum of all weights should be 1.
If the sum is > 1.0 all values will be decreased to fit this scale keeping the relative values.
E.g $[(''burrata'', 1.0), (''comte'', 1.0), (''cheddar'', 1.0)]$ will be processed as
$[(''burrata'', 0.33), (''comte'', 0.33), (''cheddar'', 0.33)]$.

Analytic tasks

A3.5

The search query of cheesy3-7fdaa098 was used to search using the TFIDF based similarity and Cosine similarity separately. Since the cosine similarity method only changes the relative weights between tokens we expect that both methods find the same websites just in a different order as very high TFIDF values can not dominate smaller ones. The following search results and scores were found.
Cosine similarity results(score):

1. http://mozzarella-and-edam.cheesy3(0.45634104313234863)
2. http://asiago-and-brie.cheesy3(0.2909463732041253)
3. http://gouda-and-muenster.cheesy3(0.2372712574597625)
4. http://rich-mozzarella.cheesy3(0.2258348907332743)
5. http://quark-and-slovakianbryndza.cheesy3(0.21560544761773803)
6. http://nutty-lancashire.cheesy3(0.15970650426488966)

TFIDF results(score):

1. http://mozzarella-and-edam.cheesy3(0.08394946401445183)
2. http://quark-and-slovakianbryndza.cheesy3(0.062292726687674126)
3. http://gouda-and-muenster.cheesy3(0.05681096673915881)
4. http://rich-mozzarella.cheesy3(0.043235134504687064)
5. http://asiago-and-brie.cheesy3(0.04012073922256978)
6. http://nutty-lancashire.cheesy3(0.029314224323611356)

These are the TFIDF values for the individual tokens per website:

url	burrata	slovakianbryndza	cantal
mozzarella-and-edam.cheesy3	0.020524	0.036060	0.027366
asiago-and-brie.cheesy3	0.008024	0.000000	0.032097
quark-and-slovakianbryndza.cheesy3	0.031146	0.000000	0.031146

Looking at the TFIDF values of the tokens we can see that http://mozzarella-and-edam.cheesy3 stays first result in both methods as the token ”slovakianbryndza” is only contained here and therefore dominates in TFIDF by adding 0.036060 to the final score for TFIDF and being the only website that contains all three tokens for the cosine similarity.
The second place already changes with http://asiago-and-brie.cheesy3 for cosine similarity and

http://quark-and-slovakianbryndza.cheesy3 for the TFIDF similarity. We can see that "cantal" dominates "burrata" in the TFIDF method for asiago-and-brie.cheesy3 pushing the result down even though both tokens are contained on both websites. The cosine similarity method lifts it up though as the vectors align better.

Aufgabe Z3.2

Looking at the formula for the cosine similarity it is obvious that if ||a||=||b||=1 $\frac{doc*query}{||a||*||b||} = doc * query$ So if we preemtively normalize the vectors we would save calculating the norm for each vector, multiplying the norms and the following division when performing the search.

Normalizing the vector can be done by dividing each dimension by the whole vectors length so for a vector $A = (a_1, a_2, a_3, ..., a_n)$ we can find the normalized new dimensions as follows $a_{i(new)} = \frac{a_i}{\sqrt{a_1^2+...+a_n^2}}$.

This functionality has been implemented in the VectorIndex.normalize() method. The normalized values will then automatically be used when using the VectorIndex.searchQueryCosine() method.

A4.3 / A5.2

Upon trying to calculate the page rank for cheesy6-54ae2b2e we first notice that the page rank algorithm does not converge as 1000 iterations are met without reaching a maximum difference below 0.0001 in page rank changes. Closer inspection shows the following loop where http://nutty-cheddar24.cheesy6 -> http://crumbly-cheddar.cheesy6 -> http://brie24.cheesy6 -> http://nutty-cheddar24.cheesy6 -> ...:

```
-------------------------------- 989 --------------------------------
Node:http://brie24.cheesy6                  pageRank:0.25
Node:http://cheddar24.cheesy6               pageRank:0.0
Node:http://crumbly-cheddar.cheesy6         pageRank:0.25
Node:http://nutty-cheddar24.cheesy6         pageRank:0.5
-------------------------------- 990 --------------------------------
Node:http://brie24.cheesy6                  pageRank:0.25
Node:http://cheddar24.cheesy6               pageRank:0.0
Node:http://crumbly-cheddar.cheesy6         pageRank:0.5
Node:http://nutty-cheddar24.cheesy6         pageRank:0.25
-------------------------------- 991 --------------------------------
Node:http://brie24.cheesy6                  pageRank:0.5
Node:http://cheddar24.cheesy6               pageRank:0.0
Node:http://crumbly-cheddar.cheesy6         pageRank:0.25
Node:http://nutty-cheddar24.cheesy6         pageRank:0.25
-------------------------------- 992 --------------------------------
Node:http://brie24.cheesy6                  pageRank:0.25
Node:http://cheddar24.cheesy6               pageRank:0.0
Node:http://crumbly-cheddar.cheesy6         pageRank:0.25
Node:http://nutty-cheddar24.cheesy6         pageRank:0.5
```

After implementing the rank source component we test again with a rank source of 0.15 and the page rank algorithm converges in 48 iterations.

```
-------------------------------- 44 --------------------------------
Node:http://brie24.cheesy6                  pageRank:0.33254940795056187
Node:http://cheddar24.cheesy6               pageRank:0.037500000000000006
Node:http://crumbly-cheddar.cheesy6         pageRank:0.30961695053007277
Node:http://nutty-cheddar24.cheesy6         pageRank:0.3203336415193655
-------------------------------- 45 --------------------------------
Node:http://brie24.cheesy6                  pageRank:0.33254940795056187
Node:http://cheddar24.cheesy6               pageRank:0.037500000000000006
Node:http://crumbly-cheddar.cheesy6         pageRank:0.3097835952914607
Node:http://nutty-cheddar24.cheesy6         pageRank:0.32016699675797755
-------------------------------- 46 --------------------------------
Node:http://brie24.cheesy6                  pageRank:0.3326910559977416
Node:http://cheddar24.cheesy6               pageRank:0.037500000000000006
Node:http://crumbly-cheddar.cheesy6         pageRank:0.30964194724428096
Node:http://nutty-cheddar24.cheesy6         pageRank:0.32016699675797755
-------------------------------- 47 --------------------------------
Node:http://brie24.cheesy6                  pageRank:0.33257065515763884
Node:http://cheddar24.cheesy6               pageRank:0.037500000000000006
Node:http://crumbly-cheddar.cheesy6         pageRank:0.30964194724428096
Node:http://nutty-cheddar24.cheesy6         pageRank:0.32028739759808034
-------------------------------- 48 --------------------------------
Node:http://brie24.cheesy6                  pageRank:0.33257065515763884
Node:http://cheddar24.cheesy6               pageRank:0.037500000000000006
Node:http://crumbly-cheddar.cheesy6         pageRank:0.3097442879583683
Node:http://nutty-cheddar24.cheesy6         pageRank:0.320185056883993
```

A closer inspection of the rank score development shows a steady decline in changes preventing the loop from playing out and allowing convergence.

A5.3

We have been given two measures for determining relevance of a website in relation to a search query. The cosine similarity determines how closely a websites content will fit the search queue and the page rank determines how relevant a website is deemed to be to people in general and therefore most likely will be the content a person might be looking for. We neither want a website to fit the search query well but be completely irrelevant nor do we want a website that is very popular but unrelated to our search query. Both factors need to play into the end result equally for a website to fit our query as well as most likely containing the content we are looking for.
Therefore for combining the scores an equal measure was implemented, first transforming each of the found pages scores onto a scale of [0.0 - 1.0] so that one score may not dominate the other and then summing normalized page rank and normalized cosine similarity scores up.
This way each score has equal power in determining the relevance on a scale of [0.0 - 2.0].