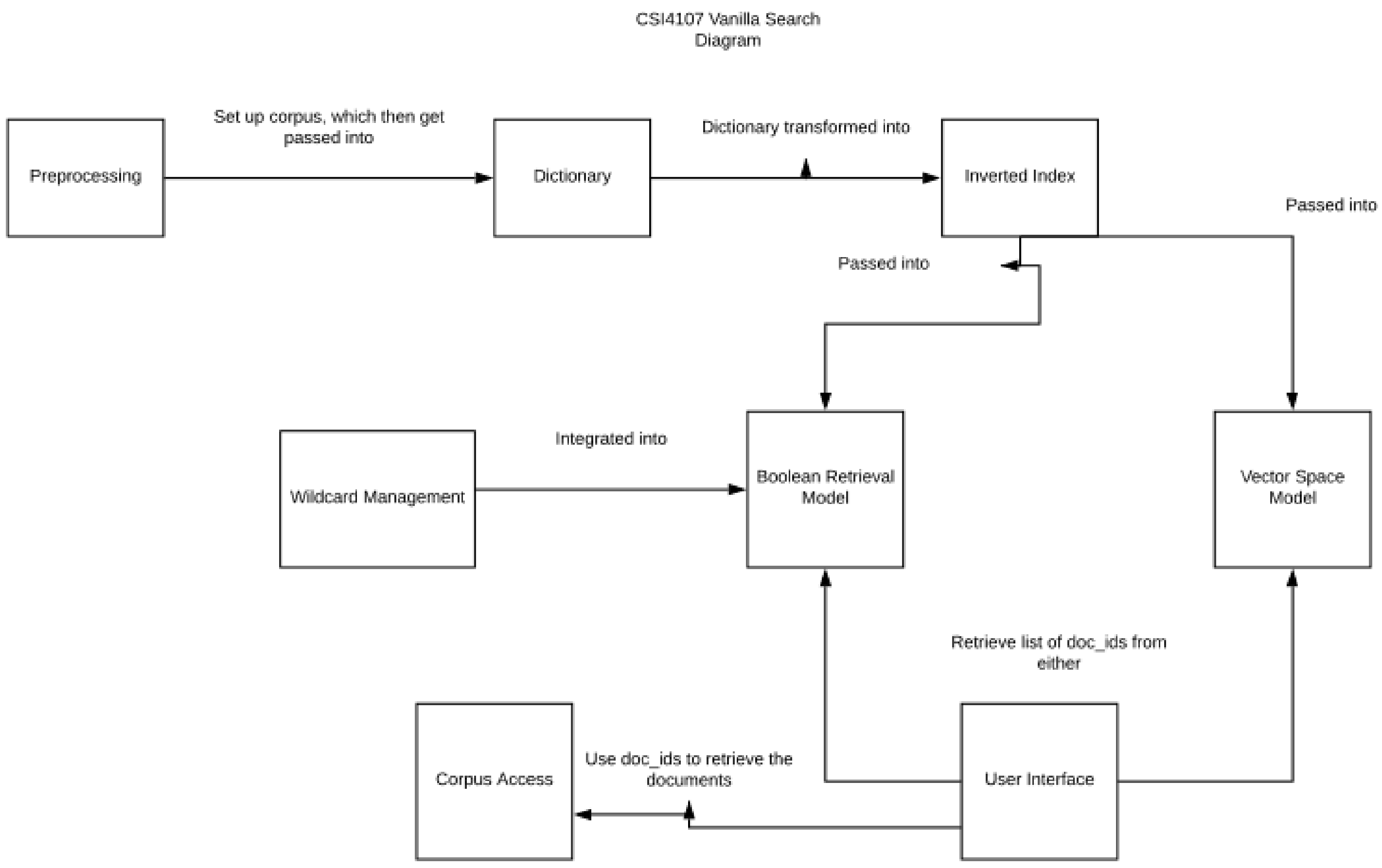
The CSI4107 Vanilla Search System

Armand Syahtama, 8253748

Vanilla Search Engine System Architecture Diagram


CSI4107 Vanilla Search Diagram
Preprocessing
Set up corpus, which then get passed into
Dictionary
Dictionary transformed into
Inverted Index
Passed into
Passed into
Wildcard Management
Integrated into
Boolean Retrieval Model
Vector Space Model
Retrieve list of doc_ids from either
Corpus Access
Use doc_ids to retrieve the documents
User Interface

Modules Included

1 - Preprocessing Module

Functionality

In the preprocessing module, I set up an abstract PrepocessingBase class with an empty abstract method, *'preprocess_collections*'. The class also contains a named tuple that represents the uniform format document structure to be followed regardless of the source of the collection.

No matter which collection we process, the uniform format of a document within the corpus contains a doc id, a title, the full text and an excerpt from the text.

I structured my preprocessing module like this, because the initial formats of the collection can vary between collections, so having a child class for each collection to handle it's own preprocessing will lead to easier to refactor code in the future and make it easier to add new collections to the corpus.

For the vanilla system, I focused on the preprocessing for the uOttawa CSI courses. It involved loading the HTML page with Python's Requests package, and then using the Beautiful Soup package to scrape through the html elements to find the text needed for the corpus. Once the scraping is complete and the documents set up, all the documents are then written to a json file that represents our corpus, which the other modules will read from for their own purposes

Limitations

None

Problems Encountered

A problem I found was trying to find the HTML elements that contained the information that I wanted. This was done by viewing the source code of the web page and sifting through the html to find the text I wanted. In this case, each course in the page was contained in a div with the class '*courseblock*'. In each *courseblock*, the title was contained in another div called *courseblocktitle* and the description in *courseblockdesc.*

Another problem was with trying to write the list of documents to JSON. The Python Json package raises an exception when you try to write a named tuple to a json file, which was what I used to store the documents. To fix that, I transformed all the named tuples into python dictionaries and then wrote that into the corpus.json file.

2 – Dictionary Module

Functionality

The dictionary module takes the corpus json file, and then sets up a dictionary that contains all the words from the corpus, including the document title and the full text of a document. The dictionary was also written to a json file. There are many modes for this dictionary, including stopword removal, Porter stemming and normalization. To accommodate all those options, the resulting dictionary is built like this:

```
{
    “unaltered”: <List of words>,
    “stopwords_removed”: <List of words>,
    “stemmed”: <List of words>,
    “normalized”: <List of words>,
    “fully_altered”: <List of words>
}
```

- The unaltered list contains all the words without any changes
- The stopwords removed list strips out all stop words from being added
- The stemmed list stems each individual word
- The normalized list normalizes the form of the word
- The fully altered list performs all three operations on the word

Limitations

None

Problems Encountered

None

3 – Inverted Index Module

Functionality

Loads both the corpus and dictionary json files and the sets up the inverted index. To build out the inverted index, iterate through each word in the dictionary, check it against every document in the corpus, and if the corpus, contains the word, count every instance of that word in either the title or the full text, and then add the set of doc ids and count to the index. I chose to record the count of each word, because I chose to represent the term frequency using the raw frequencies. Since there were 5 different modes for the dictionary, the inverted index also contains 5 different modes for each dictionary mode

Limitations

None

Problems Encountered

None

4 – Corpus Access Module

Functionality

This module loads up the corpus, then takes in a list of doc ids from the user. The module then returns the corresponding documents from the json back to the user

Limitations

None

Problems Encountered

None

5 – Boolean Retrieval Model Module

Functionality

Takes in a user query with Boolean operators and then cross references each token with the inverted index to retrieve the doc ids from the index. To make processing easier, I decided to change the notation of the query from infix to postfix at the beginning of the retrieval process. Since I chose to do Wildcard management as the optional module, the code for handling the wildcards is also within this module.

Since this is the Boolean Retrieval Model, there are set operations involved with the query

- AND: intersect between resulting sets from two tokens
- OR: union between resulting sets from two tokens
- NOT: the complement of a set from a single token

Limitations

You can't search for a document that contains those Boolean operator words. The retrieval model treats those words as operations, rather than words to query. So, if you wanted to look up 'AND' with Boolean retrieval mode, your results will come up empty.

Problems Encountered

The query "operating system" caused a problem with my implementation (or any query without Boolean operators in-between). Since there were no Boolean operators, after each set of docs per token was retrieved, my system ended up only taking the top most set of doc ids (ex. All docs with word system in it), instead of docs with both words. To solve this, I decided to treat any tokens with no Boolean operators in-between them as an implicit AND, and then intersect all the remaining sets. Semantically, an implicit AND would make sense as when a user types in those two tokens, the user would expect results containing both tokens within.

6 – Vector Space Model Module

Functionality

Performs the retrieval of documents from a user query, using the vector space model. The Weight Calculation and the Retrieval are both in the same python class. For the weight calculation, I chose to represent the Term Frequency (TF) as the raw frequency of occurring words in a document, as I was already keeping track of the number of appearances of each word. For the vector score calculations, I chose to set all the words from the user query with an equal weight of 1. The vector score was calculated using the inner product between the query vector and the document vectors

Limitations

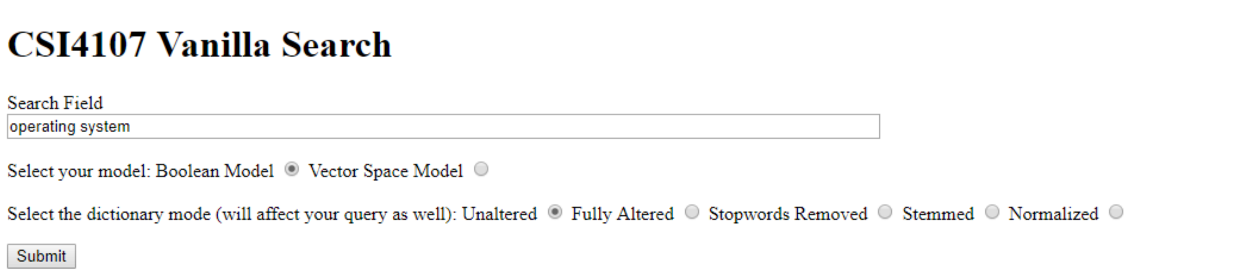
None

Problems Encountered

None

7 – User Interface Module

The user interface is a simple interface using the Flask python package to set up and render HTML pages and serve them up.



CSI4107 Vanilla Search

Search Field: operating system

Select your model: [✓ Boolean Model] Vector Space Model

Select the dictionary mode (will affect your query as well): [✓ Unaltered] Fully Altered Stopwords Removed Stemmed Normalized

The search page allows the user to switch between searching using Boolean Retrieval Model or the Vector Space Model. You can also choose which dictionary mode you want to search against as well. For instance, choosing the Stemmed option will stem the words in your query, and search against the stemmed variation of the inverted index. The results page shows all the matched docs and an excerpt line from the document (in this case, the first sentence)

Limitations

If you do a Boolean query with Stopwords Removed option on, it will also remove the stopwords from your query, effectively ruining the query and the expected results

Problems Encountered

None

Optional Module – Wildcard Management

Functionality

For the Boolean Retrieval, my system will handle the wildcards in a word regardless of its position. If a token within the query contains a wildcard (‘*’), it will build out all possible combinations of words possible, and then build out a query with all those words separated with an ‘OR’. Middle of the word wildcards were dealt with using bigram indexes, building them out and then performing a post-filter check against them to make sure they match the parts of the token.

Limitations

Due to the way the system was implemented, you can’t do the following with the wildcards:

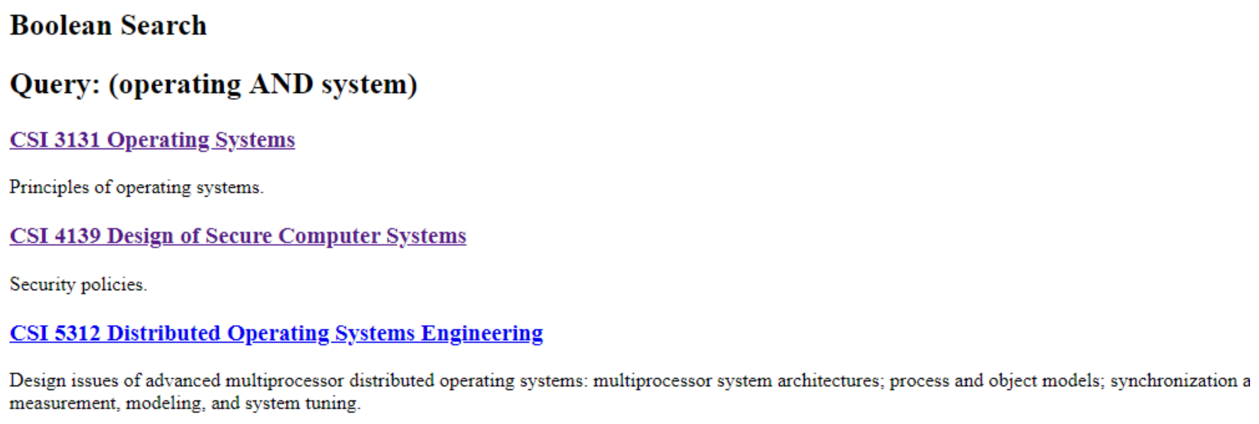
- Have a token with a wildcard at the beginning and at the end (ex: *pert*)
  What ends up happening is that, system will pick up that token starts with a wildcard and try to match the terms in the inverted index with the ending portion of it (‘pert*’). None of the terms in the inverted index have a wildcard in them, which means no results will end up being found.
- Have a token with wildcard at the beginning and at the middle (ex: *per*ting)
  Same reason as above
- Have a token with wildcard at the end and at the middle (ex: ope*at*).
  In this case, the wildcard at the end will be picked up, and the system will try to match (‘ope*at’)

Problems Encountered

Trying to handle the case where tokens have multiple wildcards in general. If there are multiple wildcards in the middle of a token, that can be dealt with easily. However, putting them in the beginning or the end causes searches to fail.

Boolean Retrieval Search Results

(operating AND system)



Boolean Search

Query: (operating AND system)

CSI 3131 Operating Systems

Principles of operating systems.

CSI 4139 Design of Secure Computer Systems

Security policies.

CSI 5312 Distributed Operating Systems Engineering

Design issues of advanced multiprocessor distributed operating systems: multiprocessor system architectures; process and object models; synchronization a
measurement, modeling, and system tuning.

(comput* AND graph*)

Vanilla System

Boolean Search

Query: (comput* AND graph*)

CSI 2101 Discrete Structures

Discrete structures as they apply to computer science, algorithm analysis and design.

CSI 4130 Computer Graphics

Interactive computer graphics.

CSI 4133 Computer Methods in Picture Processing and Analysis

Representation of digital pictures.

CSI 4140 Introduction to Parallel Computing

Models of parallel computation.

CSI 5127 Applied Computational Geometry

Design and analysis of efficient algorithms for solving geometric problems in applied fields such as Geometric Network Design, Geometric Routing and Searching.

CSI 5146 Computer Graphics

Principles and advanced techniques in rendering and modelling.

CSI 5163 Algorithm Analysis and Design

Topics of current interest in the design and analysis of computer algorithms for graph-theoretical applications; e.g.

CSI 5164 Computational Geometry

Study of design and analysis of algorithms to solve geometric problems; emphasis on applications such as robotics, graphics, and pattern recognition.

CSI 5165 Combinatorial Algorithms

Design of algorithms for solving problems that are combinatorial in nature, involving exhaustive generation, enumeration, search and optimization.

(crypto* OR security)

Vanilla System

Boolean Search

Query: (crypto* OR security)

CSI 2101 Discrete Structures

Discrete structures as they apply to computer science, algorithm analysis and design.

CSI 2501 Structures discrètes

Structures discrètes utilisées en informatique.

CSI 3130 Databases II

Advanced physical database design.

CSI 3131 Operating Systems

Principles of operating systems.

CSI 4105 Design and Analysis of Algorithms II

Theory of NP-completeness, methods for dealing with NP-complete problems.

CSI 4108 Cryptography

The notion of secure communication.

CSI 4139 Design of Secure Computer Systems

Security policies.

CSI 4505 Conception et analyse des algorithmes II

Théorie du NP-complet.

CSI 4508 Cryptographie

La notion de communication sûre.

CSI 5105 Network Security and Cryptography

Advanced methodologies selected from symmetric and public key cryptography, network security protocols and infrastructure, identification, anonymity, privacy technologies, secret-sharing, ir

(crypto* OR security) - continued

Security in encryption algorithms.

CSI 5115 Database Analysis and Design

The dimensional and multidimensional data models for data warehousing.

CSI 5116 Authentication and Software Security

Specialized topics in security including advanced authentication techniques, user interface aspects, electronic and digital signatures, security infrastructures and protocols, software vu

CSI 5128 Swarm Intelligence

Collective computation, collective action, and principles of self-organization in social agent systems.

CSI 5130 Applications Design for Mobile Devices

Programming environments for mobile devices applications: native applications and web applications.

CSI 5136 Computer Security and Usability

Design and evaluation of security and privacy software with particular attention to human factors and how interaction design impacts security.

CSI 5148 Wireless Ad Hoc Networking

Self-organized, mobile, and hybrid ad hoc networks.

CSI 5168 Digital Watermarking

Overview of recent advances in watermarking of image, video, audio, and other media.

CSI 5175 Mobile Commerce Technologies

Wireless networks support for m-commerce; m-commerce architectures and applications; mobile payment support systems; business models; mobile devices and their operating syste

CSI 5312 Distributed Operating Systems Engineering

Design issues of advanced multiprocessor distributed operating systems: multiprocessor system architectures; process and object models; synchronization and message passing primit
measurement, modeling, and system tuning.

CSI 5389 Electronic Commerce Technologies

Business models and technologies.

CSI 5789 Technologies du commerce électronique

Introduction aux modèles et technologies d'entreprise.

Vector Space Search
operating system

Vanilla System

Vector Space Search

Query: operating system

CSI 3131 Operating Systems

Principles of operating systems.
score: 9.020036653460924

CSI 5312 Distributed Operating Systems Engineering

Design issues of advanced multiprocessor distributed operating systems: multiprocessor system architectures; process and object models; synchronization and me
measurement, modeling, and system tuning.
score: 5.748064429859257

CSI 4139 Design of Secure Computer Systems

Security policies.
score: 2.8740322149296285

CSI 4141 Real Time Systems Design

Definition of real-time systems; examples.
score: 2.476092206257591

CSI 5175 Mobile Commerce Technologies

Wireless networks support for m-commerce; m-commerce architectures and applications; mobile payment support systems; business models; mobile devices and
score: 1.635986111800833

CSI 5380 Systems and Architectures for Electronic Commerce

E-commerce system architecture with a focus on relevant design patterns.
score: 1.2380461031287955

CSI 4124 Foundation of Modelling and Simulation

The modelling and simulation process from a project oriented perspective.
score: 1.2380461031287955

CSI 5134 Fault Tolerance

Hardware and software techniques for fault tolerance.
score: 1.2380461031287955

operating system – continued

CSI 5311 Distributed Databases and Transaction Processing

Principles involved in the design and implementation of distributed databases and distributed transaction processing systems.
score: 1.2380461031287955

CSI 5314 Object-Oriented Software Development

Issues in modeling and verifying quality and variability in object-oriented systems.
score: 1.2380461031287955

CSI 5122 Software Usability

Design principles and metrics for usability.
score: 1.2380461031287955

computers graphical
Vanilla System

Vector Space Search

Query: computers graphical

CSI 1390 Introduction to Computers

Computing and computers.
score: 3.5218496968182658

CSI 5149 Graphical Models and Applications

Bayesian networks, factor graphs, Markov random fields, maximum a posteriori probability (MAP) and maximum likelihood (ML) principles, e
modes algorithm, variational methods, applications.
score: 2.2380461031287955

CSI 5101 Knowledge Representation

KR is concerned with representing knowledge and using it in computers.
score: 1.7609248484091329

CSI 4140 Introduction to Parallel Computing

Models of parallel computation.
score: 1.7609248484091329

cryptographic security

Vanilla System

Vector Space Search

Query: cryptographic security

CSI 4139 Design of Secure Computer Systems

Security policies.
score: 10.919180674505572

CSI 5136 Computer Security and Usability

Design and evaluation of security and privacy software with particular attention to human factors and how interaction design impacts security.
score: 5.459590337252786

CSI 5116 Authentication and Software Security

Specialized topics in security including advanced authentication techniques, user interface aspects, electronic and digital signatures, security in
score: 4.367672269802229

CSI 4108 Cryptography

The notion of secure communication.
score: 2.2380461031287955

CSI 5106 Cryptography

Security in encryption algorithms.
score: 2.1838361349011146

CSI 5105 Network Security and Cryptography

Advanced methodologies selected from symmetric and public key cryptography, network security protocols and infrastructure, identification, a
score: 2.1838361349011146

CSI 3131 Operating Systems

Principles of operating systems.
score: 1.0919180674505573

CSI 5312 Distributed Operating Systems Engineering

Design issues of advanced multiprocessor distributed operating systems: multiprocessor system architectures; process and object models; sync
measurement, modeling, and system tuning.
score: 1.0919180674505573

Wildcard Management Showcase
c*ing

Vanilla System

Boolean Search

Query: c*ing

CSI 1306 Computing Concepts for Business

Introduction to computer-based problem solving from the perspective of the business world.

CSI 1308 Introduction to Computing Concepts

Introduction to computer based problem solving for scientific applications.

CSI 1390 Introduction to Computers

Computing and computers.

CSI 2101 Discrete Structures

Discrete structures as they apply to computer science, algorithm analysis and design.

CSI 2911 Pratique professionnelle de l'informatique / Professional Practice in Computing

Professionnalisme en informatique.

CSI 4101 Theory of Computability

Recursive functions, recursively enumerable sets, decision problems, Church-Turing thesis.

CSI 4109 Introduction to Distributed Computing

Computational models.

CSI 4140 Introduction to Parallel Computing

Models of parallel computation.

CSI 4142 Introduction to Data Science

Big data, analytics, and cloud computing; data preparation: organization, basic statistics, cleaning, and integration; data mining techniques: pattern mining, classification,

CSI 4150 Introduction to Optimization Methods

Linear optimization models and their solution.

***ity AND po*s**

Vanilla System

Boolean Search

Query: *ity AND po*s

CSI 4139 Design of Secure Computer Systems

Security policies.

Source Links for Code

Postfix Notation Code:
http://interactivepython.org/runestone/static/pythonds/BasicDS/InfixPrefixandPostfixExpressions.html

Python set update shorthand:
https://python-reference.readthedocs.io/en/latest/docs/sets/op_update.html

Regex code for counting # of occurrences of word:
https://stackoverflow.com/questions/17268958/finding-occurrences-of-a-word-in-a-string-in-python-3

Regex for checking if word occurs in piece of text:
https://stackoverflow.com/questions/5319922/python-check-if-word-is-in-a-string

Abstract Classes in Python:
https://www.python-course.eu/python3_abstract_classes.php

Web scraping in Python:
https://medium.freecodecamp.org/how-to-scrape-websites-with-python-and-beautifulsoup-5946935d93fe

Flask Set-up:
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

Flask Jinja2 Templates:
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-ii-templates

Flask Web forms:
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-iii-web-forms