DESIGN AND IMPLEMENTATION OF THE SEARCH ENGINE MODULE IN COLDS

BY

XIAOFO YU

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2018

Urbana, Illinois

Adviser:

Professor Chengxiang Zhai

ABSTRACT

This thesis describes the design and implementation of the search engine module in a novel Cloud-based Open Lab for Data Science (COLDS) system. COLDS is a general infrastructure system to support data science programming assignments on the cloud that is currently being developed at the University of Illinois at Urbana-Champaign in collaboration with Microsoft and Intel with Azure grant support from Microsoft and a gift fund support from Intel. The annotation subsystem of COLDS is responsible for helping instructors design flexible annotation tasks and straightforward annotation of data sets using search engine results. The function of the search engine module in the annotation subsystem of COLDS includes allowing instructors to upload customized data sets, building inverted index for data sets to support fast query and selecting ranking functions with customized parameters to perform query and get a ranked list of results. The thesis describes the design and implementation of the search engine module, including specifically its data set uploading and configuration procedure, indexing of data set, storage of the data set and index, and ranking and querying with selected method, parameters and data set. This thesis also describes the background, related work, challenges and future work of COLDS and its annotation subsystem.

Thanks to the TIMAN Group.
Thanks to Professor Chengxiang Zhai.
For their help, support and guidance.

ACKNOWLEDGMENTS

Thanks to my advisor Professor ChengXiang Zhai for his enormous guidance and help. It's an honor for me to be one of his students.
Thanks to the whole TIMAN group for the advice and help.
Thanks to everyone who contributed to the project with great passion and enthusiasm.
Thanks to the Department of Computer Science and University of Illinois for all the support.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION 1

CHAPTER 2 RELATED WORK 4

CHAPTER 3 COLDS AND ANNOTATION SUBSYSTEM 7
- 3.1 Overview of COLDS 7
- 3.2 Annotation Subsystem 7
- 3.3 Summary 10

CHAPTER 4 DESIGN OF SEARCH ENGINE MODULE 12
- 4.1 Requirements 12
- 4.2 Challenges 13
- 4.3 General Designs 13
- 4.4 Search Engine Designs 15

CHAPTER 5 IMPLEMENTATION OF SEARCH ENGINE MODULE 19
- 5.1 Background 19
- 5.2 Uploading Implementation 20
- 5.3 Indexing Implementation 20
- 5.4 Search Engine Implementation 21

CHAPTER 6 EVALUATION 23
- 6.1 Instructor Perspective 23
- 6.2 Annotator Perspective 27

CHAPTER 7 CONCLUSIONS AND FUTURE WORK 29

REFERENCES 31

CHAPTER 1: INTRODUCTION

With the growth of the big data produced by various different sources, there are increasingly more opportunities for applying data mining, machine learning and information retrieval techniques to turn large amount of raw data produced by human activities into usable knowledge which can improve productivity and quality of life. Those opportunities also have contributed back to the growth of the big data industry and therefore raise a huge demand for a large number of data scientists and engineers who can apply data science knowledge in real life. This huge demand requires a way of educating and training people into well-qualified engineers and scientists quickly at a low cost.

In order to meet this huge demand, there are lots of attempts made to make education available for a large audience. For instance, online education has become popular nowadays among people all over the world. Existing online education platforms, such as Coursera, Udacity and EdX, which can support Massive Open Online Courses (MOOCs), have recently emerged as an effective attempt to enable education at very large scale and comparably low costs. However, these platforms are essentially online classrooms where students watch lecture videos and work on simple assignments such as multiple choice questions which can be easily auto-graded. Unfortunately, one serious limitation of all the existing online education platforms is that they cannot support online programming assignments, which are major components in various areas of computer science education, particularly data science education. This limitation has prevented students of those online courses from gaining practical experiences in programming.
It is essential to provide meaningful programming assignments to students in order to provide high quality data science education since skills for implementing an algorithm and running experiments with algorithms are necessary skills for any data scientist and engineer. Ideally, meaningful programming assignments in data science education should involve real world data sets. Unfortunately, such real-world data sets can be very large and sensitive. Therefore, those real world data sets are often unfeasible to be downloaded by students due to size or confidentiality.

To address this problem, the Text Information Management and Analysis (TIMAN) group at the University of Illinois at Urbana-Champaign has been developing a novel Cloud-based Open Lab for Data Science (COLDS) with support of an Azure grant from Microsoft and a gift fund from Intel to enable students to work on programming assignments involving big

data sets on the cloud. The basic idea of COLDS is to deploy software tool-kits which contain algorithms for analyzing big data on a cloud-computing infrastructure. This way, students can directly experiment with different algorithms and parameters without downloading big data and the experiment results can be evaluated and compared using a cloud-based leaderboard.

From the industry perspective, there are also benefits from contributing data sets to COLDS such as visibility, training candidates with relevant skills, and get annotations of their data sets via crowd-sourcing. This last benefit would also create new research opportunities in the sense that new data sets can be used by researchers to study new data science tasks. To deliver this benefit, we have designed and implemented a general Annotation System for COLDS.

Using the Annotation System, educators and instructors can assign annotation tasks using their own data and specifying characteristics of the search method they want to use. Students can easily access the tasks assigned to them or use a customized query to fetch data and make annotations. Since new data sets that have no previous annotations can be annotated using such an Annotation System, the Annotation System can also lead to the creation of new test set to enable new research in data science.
Architecture-wise, the Annotation System is a novel subsystem in COLDS that includes user-interface module[1], Database module[2], Annotation module[3] and Search Engine module. The interface module provides an easy-access interface for instructors, students and researchers to use the functions provided by the Annotation and Search Engine module. The Database module provides database management for annotation and query data generated by users. The annotation module simply provides the function of making annotations given a query and top k documents retrieved by the search engine.

This thesis presents the design and implementation of the Search Engine module of the Annotation Subsystem. The Search Engine module plays an important role in the annotation subsystem and provides many useful functions such as data set storage, index building, ranker and parameter specification, and ranking documents given a query. In order to realize those functionality, we designed the Search Engine module to accept data sets from user and store data sets on disk. Once the data sets are uploaded, the Search Engine module would automatically build an inverted index based on the configuration provided by the user and execute queries from users at run time. The Search Engine module interacts with the Database module when it accepts a new data set uploaded from the user to store and index

it, and interacts with Annotation module or user-interface module when there is a query with any ranker and parameter.

The Search Engine module enabled the Annotation Subsystem to support storing and indexing customized data sets, quick queries using different ranking algorithms and parameters, and flexible design of annotation assignments. This thesis will provide an overview of the design of the Annotation Subsystem and the requirements for the Search Engine module, discuss the challenges in designing and implementing the Search Engine module and how we address them, and present the details of the design and implementation of the Search Engine module.

CHAPTER 2: RELATED WORK

Three previous theses by Li [1], Liu [2], and Wei [3] described three other components of the COLDS system, i.e., the annotation module, the database module, and the instructor-annotator module, respectively. This thesis describes the search engine module; together with those three previous theses, the four theses complete the description of the COLDS system.

The three previous theses have already provided a general discussion of related work to big data education, which we also include here for completeness. The opportunity of leveraging large amounts of data (i.e., "big data") in all kinds of application domains has been recognized by the US government for several years now[4]. In the past a few years, much progress has been made in big data research as shown by the creation of new conferences dedicated to this topic such as the IEEE Big Data Conference [1]. On the education side, many universities have created new courses on Data Science; for example, a search with "data science" as a query on Coursera (https://www.coursera.org/) returns a large number of online courses in the general area of Data Science. However, a significant challenge in these online courses is how to enable students to work on meaningful large-scale programming assignments involving large data sets. There have been a few systems that can help support online programming assignments in Data Science. One of them is the MLComp[5], which supports some machine learning experiments with all the experiment details documented in the system. However, the system does not support grading of programming assignments, or enable very large data sets to be used. It also does not support annotations of data sets as COLDS does. Another similar effort is Virtual Information Retrieval Lab (VIRLAB)[6], which is a web-based interactive tool that enables easy implementation of retrieval functions unlike existing command line based IR toolkits. However, this system does not support annotation tasks of data sets and does not have the power of creation of large data sets via crowd-sourcing.

Creation of annotated data sets is required in order to evaluate algorithms in Data Science. However, annotations generally require much manual labor, thus is expensive. This limited the availability of annotated data sets that can be used for supporting new research in data science. Recently, crowd-sourcing annotations of data sets has become very popular due to its affordability[7, 8, 9]. However, existing methods mostly rely on paying many cheap

[1] http://cci.drexel.edu/bigdata/bigdata2017/index.html

labors (e.g., using Amazon Mechanical Turk (https://www.mturk.com/)). In contrast, the Annotation Subsystem of COLDS enables a novel way of crowd-sourcing annotations of potentially very large data sets by leveraging student assignments where a large data set can be distributed among many students to create annotations and all the annotations can then be aggregated to form a large data set. This novel strategy potentially enables creation of annotated data sets without any cost since the students would learn evaluation skills from working on such annotation assignments. Such an assignment-based annotation strategy has proven effective when used in the Text Retrieval and Search Engines course on Coursera (https://www.coursera.org/learn/text-retrieval).

The Annotation Subsystem supported by the work in this thesis currently supports annotations for search engine evaluation. The evaluation methodology supported by this system is based on the Cranfield Evaluation methodology [10], also called test-collection evaluation [11]. In such an approach, the main challenge is to create a test collection consisting of three parts: sample queries, sample documents, and relevance judgments. Collecting documents is relatively easy, but collecting queries and relevance judgments can be a challenging task since it involves user effort. The traditional approaches to solving this problem rely on paying users to make annotations or running evaluation competitions as done in TREC [12]. The Annotation Subsystem of COLDS provides a more scalable way to solve this problem, which would enable the creation of potentially many new data sets for evaluating search engines.

In addition, the Annotation Subsystem also employs the power of crowd-sourcing[13] from reliable sources, i.e. students who are eager to learn valuable data science skills. Moreover, students typically have the motivation to participate in annotation collection tasks and have a relatively serious attitude toward those tasks as those annotations will be used for creating meaningful data sets that can benefit students. Similar approaches can also be found[14] which shows promising potentials of collecting relevance judgments efficiently and reliably in this way.

There are also many existing tool-kits that implement data mining, machine learning and information retrieval algorithms and provide interfaces for using them[15, 16, 17]. The Stanford-NLP-took-kit[2] provides a list of comprehensive natural language processing algorithms that can be used either in applications or research. The CogComp-NLP tool-kit[3] is

[2]https://github.com/stanfordnlp
[3]http://nlp.cogcomp.org/

also a Java based natural language processing took-kit which also provides flexible interfaces for researchers and application builders. On the other hand, information retrieval tool-kits[4] also play important roles of assisting people to experiment and apply information retrieval algorithms. They are easily portable and compatible in many different environments and suitable for various different tasks. The designs and implementations of these tools are very useful and inspiring for the design and implementation of the Search Engine module in the Annotation Subsystem since the general requirements for the Search Engine module are very similar to those of these tool-kits. Moreover, these tool-kits are also very useful references for the Search Engine module and they can even be incorporated into the Search Engine module if necessary in the future.

[4]https://www.lemurproject.org/

CHAPTER 3: COLDS AND ANNOTATION SUBSYSTEM

In this chapter, we briefly introduce the whole COLDS system and describe the four major components of its Annotation Subsystem; the Search Engine module is one of them.

3.1 OVERVIEW OF COLDS

Cloud-based Open Lab for Data Science (COLDS) is a general system that enables students to experiment with various algorithms for processing big data for massive online data education. There are many different subsystems in COLDS which contribute different features that allow students to implement their own algorithms, tune parameters and experiment on big data provided in the back-end of COLDS. The Annotation Subsystem in COLDS is one that provides annotation and assignment capabilities of the many different systems in COLDS.

3.2 ANNOTATION SUBSYSTEM

The Annotation Subsystem in COLDS can be viewed as an independent system that allows students to complete annotation assignments via a web interface. There are three different modules in the subsystem that works together in the back-end, that are Annotation module[3], Database module[2] and Search Engine module. The user-interface[1] is separated into Instructor module and Annotator module which bridges two group of users to the other three core modules. The overall architecture of the system is illustrated in the following sections as well as in Figure 3.1.

The back-end of the system in written in Python and Flask[1] to build a REST API interface for all kinds of front-end such as web and mobiles. The current front-end is also supported by the Flask framework to enable simple user interactions with the back-end.

The Database module is the most basic module that handles data reading and writing for other modules in the system. It is a core module in the entire Annotation Subsystem not only because it is responsible for storing all sorts of information correctly, but also because it has to be well designed so that every read and write operation required from other modules and users can be done efficiently. Briefly speaking, the Database module is one of the key

[1] http://flask.pocoo.org/

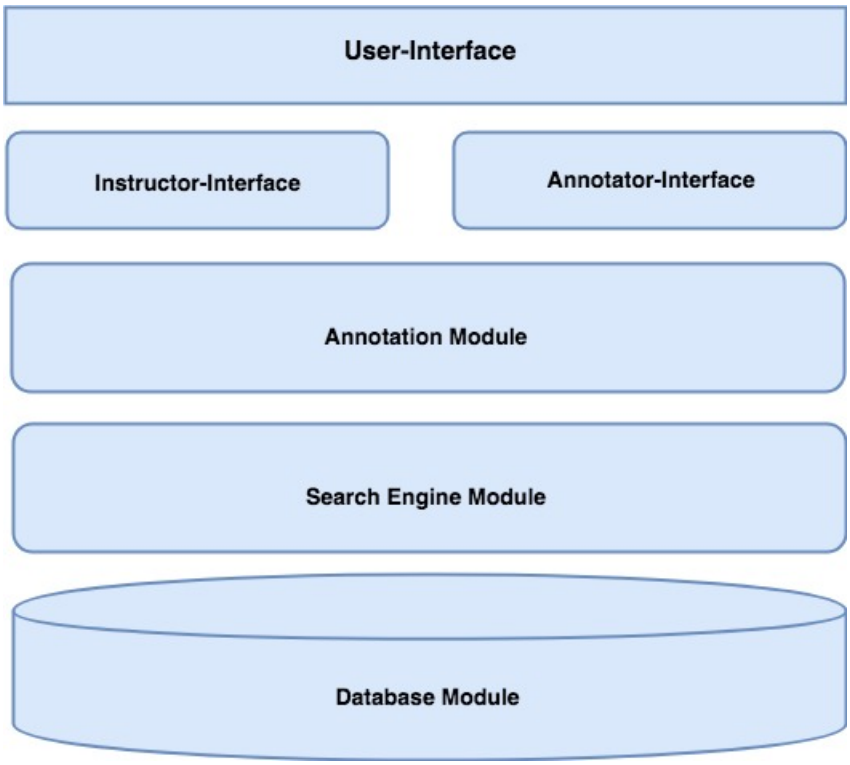

Figure 3.1: Architecture of Annotation Subsystem

modules that can make it possible for the Annotation Subsystem to scale up and scale out in the future when substantial amount of users exploit the system for collecting large number of annotations on huge data sets. In detail, Database module records annotations made by students, assignments made by instructors, and useful information about data sets created by instructors from the Search Engine module.

The Annotation module is responsible for assignment creation by instructors and handling annotations made by students. Instructors can create assignments of annotations for students using the Annotation Subsystem. An assignment typically include several queries on a data set and an algorithm with specified parameters. The Annotation module will call the Search Engine module to execute those queries using the specified algorithm with parameters to fetch the ranked lists of documents in the data set and then give the results to students. Once students finish the annotation tasks, Annotation module will call the Database Module to store students' annotations.

The Search Engine module is responsible for creating and indexing data sets, and executing queries with specified algorithm and parameters. It interacts with the Database module

when instructors upload new data sets to the Annotation Subsystem to store key information of the new data sets. Moreover, it uses the information stored by the Database module to locate documents or data sets when there is a need to retrieve them. For example, the Search Engine module will need to load the index of a data set when executing search engine algorithms and the location of the index can be found using the information from the Database module. It also provides interface to the Annotation module when instructors create new annotation assignments, and directly responds to the user-interface when students use it as a search engine for an arbitrary data set, query, or algorithm.

3.2.1 Database Module

This section briefly describes the design and implementation of the Database module. The Database module is built upon a NoSQL database called MongoDB[2] in favor of its efficiency, flexibility and scalability. The Database module is responsible for storing all types of information including user information, data set information, assignment information and annotation information. Currently, the Database module includes User, Assignment, Annotation, Query, Document, and Data set collections which store application data safely at the back-end. Flexibility of MongoDB enables developers to change or amend existing schema and collections easily so that it is easy to accommodate constantly changing user needs in the future.

However, a lot of application data requires low latency retrieval. High cost disk read in the database is not suitable for such data. Therefore, the Database module utilize an in memory key-value store called Redis[3] to make fast read and write of such data possible in the system. For example, user tokens are used for authorization and authentication of different actions. It is extremely slow for the server to seek disk for validation of the token whenever user takes an action.

3.2.2 Annotation Module

Annotation Module is the interface of the back-end because it directly provides APIs to the front-end for the users. There are two different groups of users in the system, that

[2]https://www.mongodb.com/
[3]https://redis.io/

are instructors and annotators. Instructors can use the annotation module to create new assignment of annotations. Assignment typically contains many queries on a given data set and a retrieval algorithm with specified parameters. Once the instructor confirms those parameters as he/she needs and assigns the assignment to annotators, the Annotation module will call the Search Engine module to execute and return a ranked list of documents for annotation. Then, the annotators will be able to annotate the relevance of query and documents. The Annotation module will be storing those annotations using the Database module.

3.2.3 Search Engine Module

The Search Engine module is also built in Python on top of MeTA[4] by using the Python bindings as an interface. The design and implementation details will be covered in the next chapters for the Search Engine module.

3.2.4 User Interface

The user interface is built with common web programming languages such as HTML, CSS and Javascript as well as modern frameworks such as Bootstrap[5] and JQuery[6]. It provides basic interfaces for users to register, log in and modify profile information. The instructor interface is designed for instructors to perform actions such as uploading data sets, creating annotation assignments and viewing annotations. The annotator interface is designed for annotators to perform actions such as searching on data sets, annotating documents, and going through their annotation assignments.

3.3 SUMMARY

In this chapter, we provided a general overview of COLDS as well as its Annotation Subsystem. It is clear to see in this chapter that the Annotation Subsystem can benefit COLDS by providing the ability of collecting quality annotations of any new data sets.

[4]https://meta-toolkit.org/
[5]https://getbootstrap.com/
[6]https://jquery.com/

Moreover, this chapter also introduces the architecture of the Annotation Subsystem with four very important modules that are responsible for different aspects of the Annotation Subsystem. Therefore, it is not hard to see the importance of including the four modules in the Annotation Subsystem in order to make the whole system easy to use for instructors, students and researchers. In the next chapter, we will discuss the design of the Search Engine Module, which is the focus of this thesis.

CHAPTER 4: DESIGN OF SEARCH ENGINE MODULE

This chapter describes the design of the Search Engine module as well as some key design choices made in the whole Annotation Subsystem for the purpose of effective annotation collecting. The Search Engine module is built upon MeTA tool-kit[18] to exploit its powerful implementations of various Information Retrieval and Natural Language Processing algorithms.

4.1 REQUIREMENTS

Among many detailed and specific requirements for the Search Engine module, uploading, indexing and searching documents in data sets are the most basic and general but important ones.

4.1.1 Data Uploading

The Search Engine module needs to support data set uploading from instructors so that it can be possible for instructors to publish new data sets and collect annotations from annotators. The Search Engine module should take files transferred from the user-interface via web-protocols because the whole system is web based and users perform all actions through the web-interface.

4.1.2 Data Indexing

The Search Engine module needs to query the data set with different algorithms, parameters and query words. Majority of existing algorithms benefit from having an index structure of the data set such as Inverted Index to speed up the computation because the index can immediately answer questions for algorithms as a table look up.

4.1.3 Search

The major functionality of the Search Engine module is that it needs to fetch a ranked list of documents given a query and specified algorithm and its parameters. For example, a user might use the search engine to try some random queries on his/her own customized data set to experiment which query can fetch enough number of relevant documents. This requirement can be viewed as a common feature of most existing search engines on the web.

Moreover, the Search Engine module should also perform a lot of similar tasks at the same time because the annotation assignments usually contain many different queries.

4.2 CHALLENGES

There are many challenges to satisfy all the requirements and needs for the Search Engine module. This section briefly presents major challenges of realizing the features of data uploading, indexing and searching.

First of all, the Search Engine module needs to implement many different algorithms with many different parameters. So the Search Engine module needs to have as many different implementations as possible and it's important to invoke them whenever the request from users comes. Furthermore, it is more clear and intuitive to provide an easy to understand and general interface to the user-interface and other modules in the system to make the Search Engine module and the whole system as scalable as possible.

As the requirements state, users are able to customize and upload their own data sets for annotations with arbitrary algorithms and parameters. Therefore, the Search Engine module has to take care of accepting uploaded data and organizing the data in a way that is scalable when there are many more users and data sets and when the data sets can be big in size.

On the other hand, in order to speed up the search and query tasks, it is often important to compute and store the index beforehand. So another challenge is to organize the index data with the consideration of execution speeds which affects latency of user queries and throughput of massive annotation assignments which require lots of queries done by the Search Engine module at the same time.

4.3 GENERAL DESIGNS

In this section, we will discuss the important design choices and details of the entire Annotation Subsystem that are related to the specific designs of the Search Engine module. Generally, the relationship among collections in the Database module is very useful since it is more efficient for the Database Module to store information that takes long time to generate or compute.

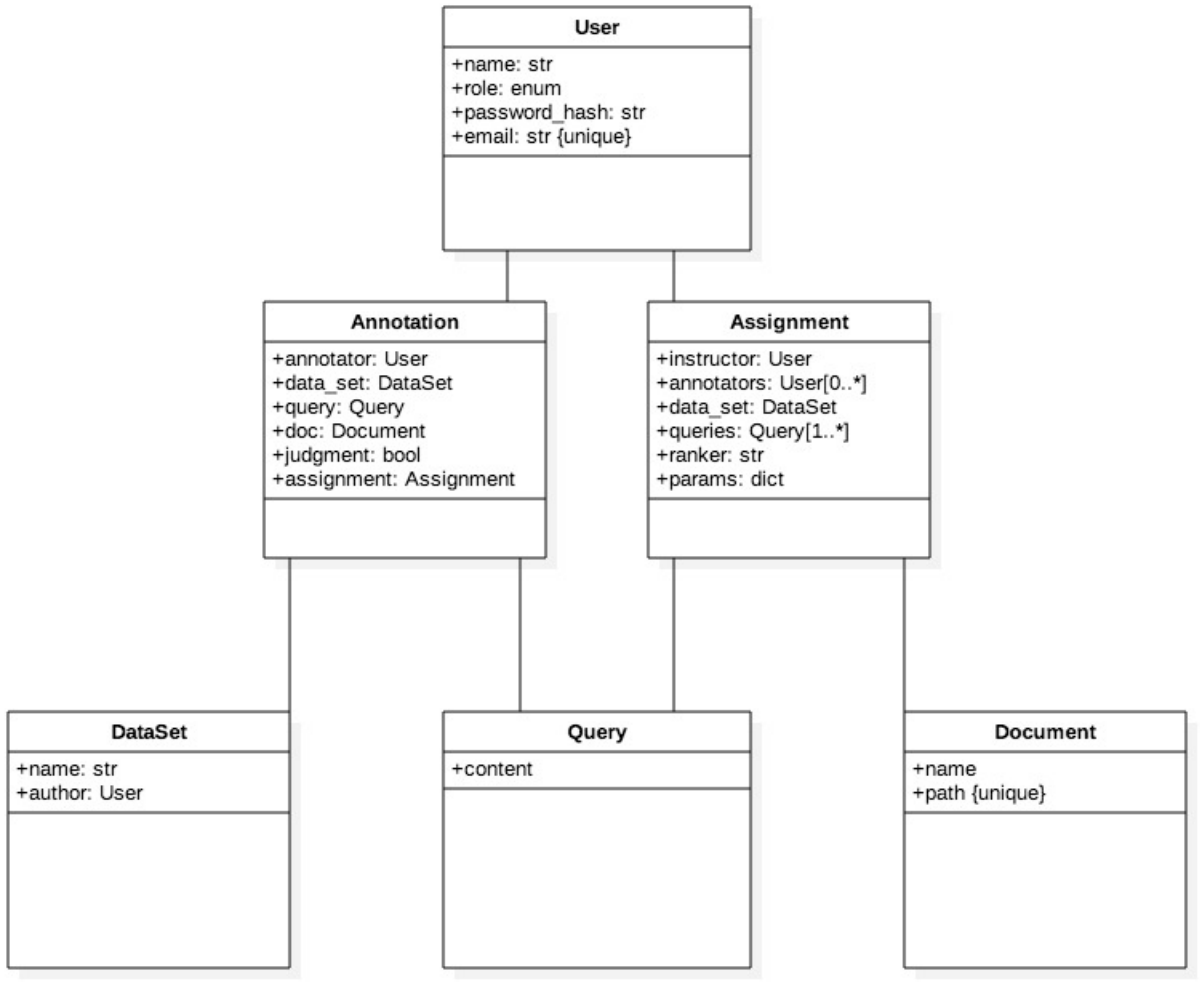

Figure 4.1: Database Module Relationship Diagram

As shown in Figure 4.1, there are complicated relationships among the Assignment, Annotation, DataSet, Query and Document collections. However, these complex relationships enable both the Annotation and Search Engine module to store information about assignments, data sets, queries and documents in a safe place. Specifically, the Search Engine module can effectively store lots of useful document information after indexing a new data set uploaded by instructors.

4.4 SEARCH ENGINE DESIGNS

In this section, we will discuss the important design choices and details of the Search Engine module. Many of the design choices aim to solve the challenges of efficiently storing and querying the data. There are two major components of the Search Engine module. They are the upload and index component and the search component respectively. Both of them are built upon MeTA[18]. This tool-kit is chosen for its powerful and comprehensive implementations of various IR algorithms. Experiments show that it has relatively similar performance compared to well-known tool-kits such as Lucene[1] but it is also able to carry out computation in parallel. The design of these two components, indexing and searching, will be discussed in the following sections.

4.4.1 Data Uploading and Indexing

It is important for users to be able to upload their own customized data sets and collect annotations for the documents inside. The Search Engine module accepts files using form data transferred from the frontend so that uploading becomes possible. Since users especially those who want to publish new data sets may grow unexpectedly and the number of data sets can also grow rapidly, the data uploading component must be robust and scalable. Another challenge that needs to be taken into consideration is that indexing of data sets is also required for speed processing of queries. Therefore, the major design questions for this component are how to store the data, when to compute the index and how to store the index in the system now and in future.

In order to store and manage user uploaded data sets in an organized manner, the system needs a way to form a structure for the data sets. So the Search Engine module develops a way of using user name and data set together as an identifier to store and search for data set locations. Specifically, data sets can be organized in a way that all data sets created by the same author will sit in the **author/** directory and all files in the same data set will be in the **author/dsname** directory. For example, two instructors will have two directories to store their own data sets as shown in Figure 4.2. The major advantage of maintaining this structure is that it is easy to partition the data sets either in units of authors or units of data sets in the future when there is a huge demand for disk space due to the growing number and size of data sets.

[1] http://lucene.apache.org/

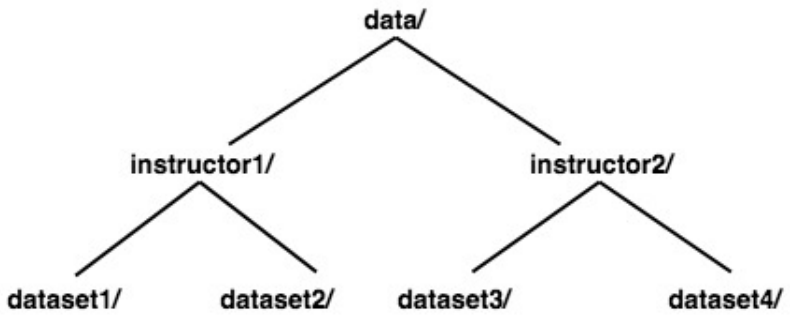

Figure 4.2: Example File System Architecture of Data Sets

On the other hand, this type of structure makes it easy to store index, too. The Search Engine module uses the same mechanism to store the index of data sets computed by MeTA. The Search Engine module specifically requires the uploaded data set to be in File Corpus format so that they can be indexed using algorithms supported by MeTA. In detail, this format makes each document in the data set resident in its own file. In addition, the instructor who uploads his own data set needs to write configuration files to tell Search Engine module how to index this data set. The first file is **file.toml** which specifies the type of corpus that's going to be indexed. The second file is **dataset-full-corpus.txt** which contains a label which can be none or each document followed by the path to the file on disk. The last file is **metadata.dat** which contains the relative path to the file on disk and the name of the document seperated by a space on each line. This format of configuration is supported by MeTA to effectively build index and search documents later on.

However, if the data set uploaded is very large, it will take a lot of time to index the data set. The Search Engine module makes it possible to first check the validity of the configurations and then perform indexing using MeTA in the background. Therefore, it is possible to exploit MeTA's parallel computing capabilities to indexing multiple data sets too.
The Search Engine module also supports other operations such as deleting the index of a data set and deleting an entire data set. It is sometimes necessary to do so because some data sets may get enough annotations but they still eat up a lot of storage space in the system.

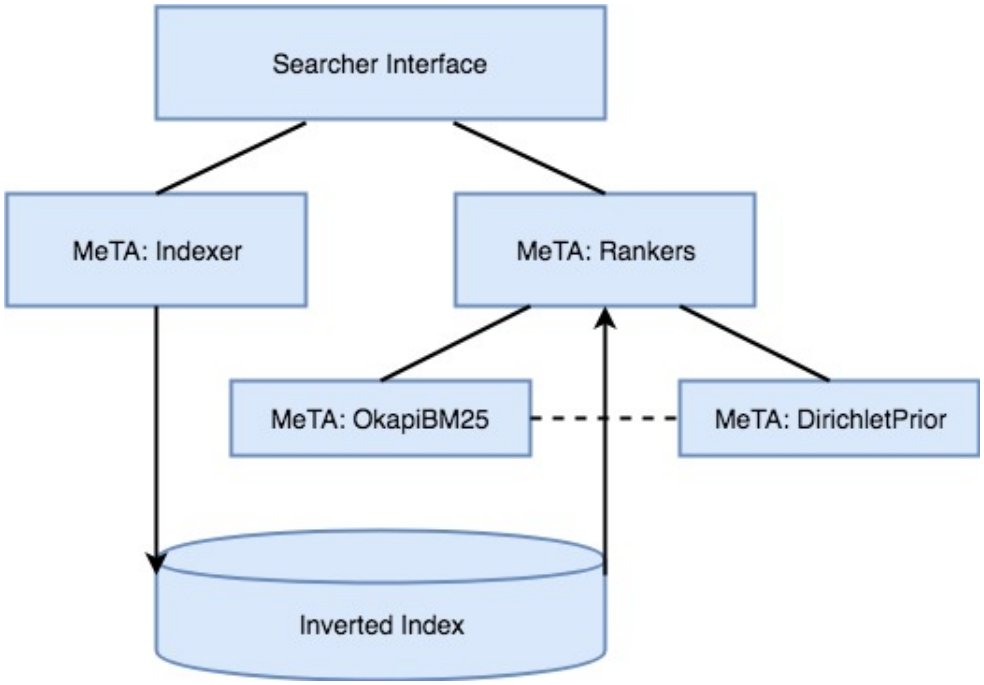

Figure 4.3: Structure of the Search Engine Module

4.4.2 Search API

The most important part of the Search Engine module is the search API that provides the ability for users to search the data set using any query with different algorithms and parameters. Fortunately, the indexing step and the MeTA tool-kit[18] makes it easy for the Search Engine module to apply many implementations of information retrieval algorithms on data sets created by users. For example, the Search Engine module allows users to choose classic algorithms such as OkapiBM25, Dirichlet Prior and Jelinek Mercer[19] and experiment with different parameter settings of these algorithms.

Since the Search Engine module is required to support different algorithms and parameters, it is clearly better to have a general and flexible interface to accept specifications of those parameters from the user side. Thus, these requirements and limitations lead to the design of a general wrapper searcher class which takes algorithm name and parameter settings as input and performs the search tasks. In the core of this searcher wrapper, it invokes different implementations of search engine algorithms and set the appropriate parameters according to user requirements.

In reality, whenever a request for searching a data set comes, the searcher wrapper first

communicates with the Database module to validate the data set. Then, the searcher will try to load the index of the required data set. In the case of missing index, the searcher will invoke the indexing algorithm again to produce the index for the data set. If the index is ready, the wrapper will validate the algorithm and parameters. In the case of invalid algorithm choice and parameter settings, the wrapper will choose the default setting and output a warning. In the end, the wrapper will invoke a real search algorithm instance to score and rank documents given the query. The results returned will include time consumed for completion, a ranked list of document name, path and score given by the ranker. In conclusion, the architecture of the Search Engine module as illustrated in Figure 4.3 is implemented in such a way that meets the requirements for the Annotation Subsystem.

CHAPTER 5: IMPLEMENTATION OF SEARCH ENGINE MODULE

This chapter describes the implementation details of the Annotation Subsystem, especially the Search Engine module. The implementation follows the design principles mentioned in the previous chapter so that the expectations and requirements can be met and the challenges can be solved effectively.

5.1 BACKGROUND

As mentioned in the previous chapter, the back-end of the Annotation Subsystem is built on top of Flask. Specifically, it uses Flask-RESTful[1] to provide REST APIs to the user interface part of the system. Therefore, the typical interface provided by the back-end can support general HTTP requests including POST, PUT, GET, and DELETE. Different URL endpoints are typically used to route requests to their service destinations. The example of concrete implementation of routing is illustrated in Figure 5.1. Majority of the request parameters reside in the body of the request as a JSON[2] while there are a few exceptions like tokens typically reside in the header of the request. This style of implementation makes it easy for developers of the Annotation Subsystem to extend existing features and add new modules to the system.

```
api.add_resource(IndexAPI, '/index')
api.add_resource(SearchAPI, '/search/<string:author>/<string:ds_name>')
api.add_resource(AnnotationAPI, '/annotation')
api.add_resource(UploadAPI, '/upload')

api.add_resource(UserAPI, '/register')
api.add_resource(LoginAPI, '/login')
api.add_resource(LogoutAPI, '/logout')

api.add_resource(AssignAPI, '/assign')
api.add_resource(AssignmentAPI, '/assignment/<string:instructor_name>/<string:assignment_name>')
api.add_resource(AssignmentUpdateAPI, '/assignment_update')
```

Figure 5.1: Example URL Routing Implementation

[1]https://flask-restful.readthedocs.io
[2]https://www.json.org/

5.2 UPLOADING IMPLEMENTATION

This section will describe the implementation details about the uploading implementation done in the Search Engine module which enables users especially instructors to upload their own data sets to the Annotation Subsystem.

The implementation of the interface of the uploading feature includes two parts, GET request handler and POST request handler. This interface can be reached from the **/upload** URL endpoint provided by the back-end.

The GET request handler is simply responsible for returning a view of the upload page back to the caller. The type of the response body is HTML. Many of the GET request handler serve the same purpose in the Annotation Subsystem.

The POST request handler is primarily responsible for accepting uploaded files from the user-interface and store them using the Database module. The request body type it takes is form with parameters like **author** and **dsname**. As described in the previous section about the design of the Search Engine module, those parameters are used to determine where to store the files of the data set. Currently, the implementation checks the file names and only allows extensions in **.txt**, **.toml** and **.dat** for security reasons. The uploading implementation also checks the validity of the request so that it can make sure the uploaded data set has at least the required files.

5.3 INDEXING IMPLEMENTATION

This section will describe the implementation details about the indexing of data sets in the Search Engine module. The actual implementation of indexing takes place in the general Search Engine wrapper described in the previous chapter about the design of the Search Engine module. However, there is no real interface implemented and designed for the users to invoke the indexing procedure. Rather, the Search Engine module itself controls the execution of indexing.

The search engine wrapper utilizes MeTA's indexing algorithms to build inverted index for data sets. There are many different types of corpus that can be indexed by MeTA with

tutorials of writing configuration files online[3]. The current implementation in the Search Engine module primarily supports the file corpus format because it is the most general format meeting the requirements of the whole system. For example, it is easier and quicker for the system to respond to a request of viewing contents of a document using file corpus than using line corpus.

As described in the previous chapter, the Search Engine module will read **file.toml**, **dataset-full-corpus.txt**, and **metadata.dat** to sort out the type of corpus, document names and other information. Then, the Search Engine module will generate another **toml** file as shown in Figure 5.2 for the configuration of indexing. In this configuration file, it tells MeTA about the analyzers and filters to be used. Typically, the unigram language model is sufficient and most commonly used in the context of Annotation Subsystem. In the end, the last step is to use MeTA's indexing algorithm to generate the index.

```
index = "testdata-idx"
prefix = "."
corpus = "file.toml"
dataset = "testdata"

[[analyzers]]
ngram = 1
method = "ngram-word"

[[analyzers.filter]]
type = "icu-tokenizer"

[[analyzers.filter]]
type = "lowercase"
```

Figure 5.2: Example Index Configuration

5.4 SEARCH ENGINE IMPLEMENTATION

As described in the previous chapter, the Search Engine module used a wrapper search engine to act as a general searcher which actually invokes algorithms implemented in MeTA. This section will focus on the implementation that follows the design principles of making the Search Engine module as general as possible.

The interface of the search engine should be flexible so that it can take any algorithm with any set of parameters. In order to make it happen, the implementation of Search Engine

[3]https://meta-toolkit.org/overview-tutorial.html

module needs to be able to find the names of algorithms and names of parameters for every algorithm in MeTA's implementations. What's more, the implementation needs to be able to make it possible to get the corresponding algorithm by name and pass the parameters into the algorithm as arguments. Therefore, such implementation can greatly simplify the interface and allow simple but general request such as shown in Figure 5.3.

```
{
  "query":"test query",
  "ranker":"OkapiBM25",
  "num_results":"2",
  "params":{
    "k1":2,
    "b":1,
    "k3":1
  }
}
```

Figure 5.3: Example Search Request

CHAPTER 6: EVALUATION

This chapter describes how users can use the Annotation Subsystem and the Search Engine module to achieve their goals. While the general flow of using the Annotation Subsystem has been shown in a previous thesis [1], this chapter will show similar evaluations that are specifically related to the Search Engine module.

6.1 INSTRUCTOR PERSPECTIVE

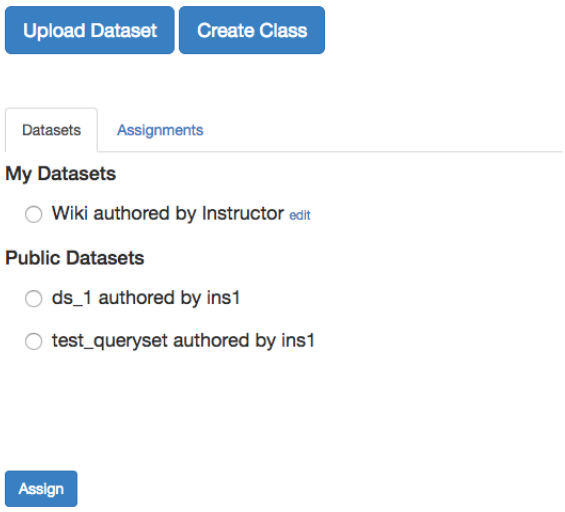

Figure 6.1: Example Instructor Home Page

In this section, we will show the interface of how the Annotation Subsystem especially the Search Engine module can be used from instructors' perspective. Once the user successfully registers as an instructor and logs into the system, there will be an instructor page, as illustrated in Figure 6.1 which allows instructors to take various actions such as uploading data sets and creating annotation assignments. In this interface, data sets that are available to the instructor are shown under either "My Datasets" and "Public Datasets". For example, under "Public Datasets", there are two data sets authored by instructor "ins1". The instructor can select one of his own or public avaiable data sets for annotation assignments.

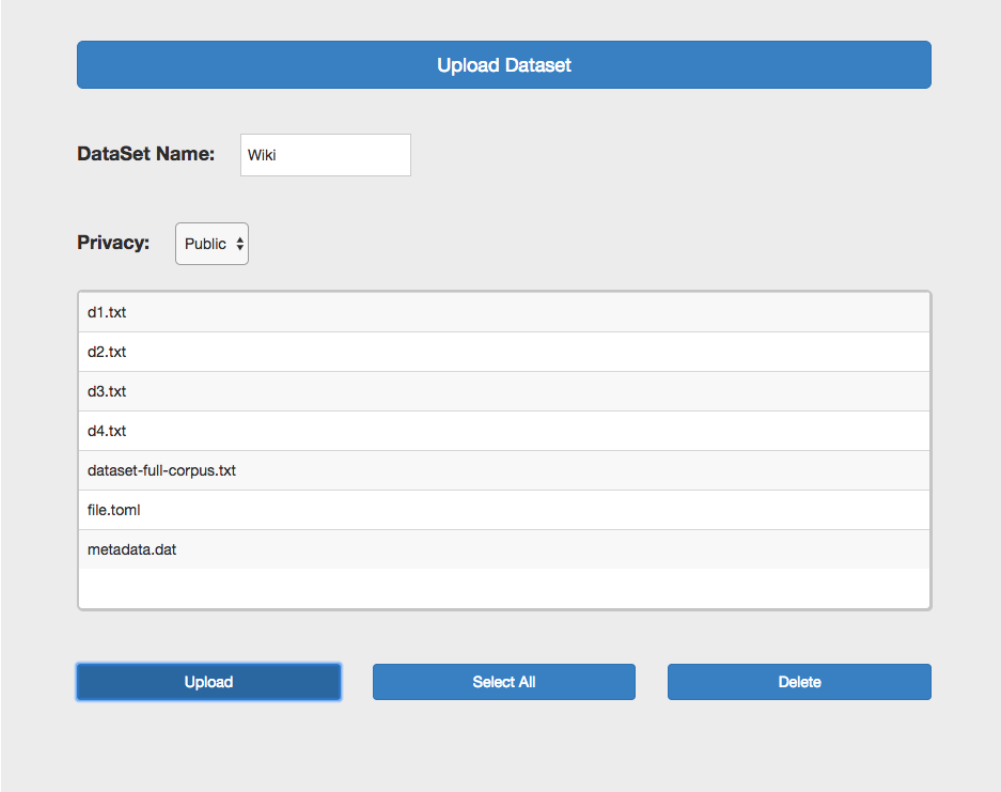

Figure 6.2: Example Upload Data Set Page

When instructors choose to upload a new data set, they will be directed to the upload data set page as illustrated in Figure 6.2. Instructors can specify data set name and whether they want the data set to be public. As shown in this interface, the instructor is creating a new data set named "Wiki" and making it public. The files that are going to be uploaded are also shown in the interface. Once they click the upload button, files will be sent to the Search Engine module. The Search Engine module will take care of validation, storing and indexing.

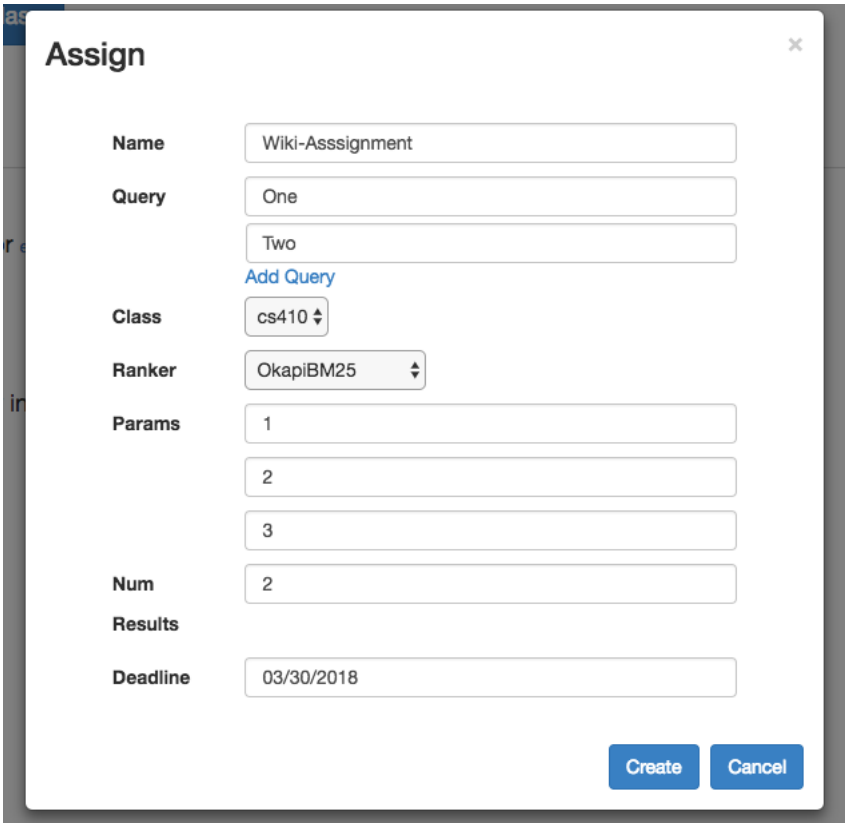

Figure 6.3: Example Assignment Creation Page

Instructors can use the instructor page to create annotation assignments as illustrated in Figure 6.3. As shown in the interface, the instructor is creating an assignment named "Wiki-Assignment" with ranker "OkapiBM25" and corresponding parameters.They can add a large numbers of queries for which they want annotations and specify algorithms and parameters. The Search Engine module will be responsible for actually scoring and ranking documents using the algorithms and parameters for every query.

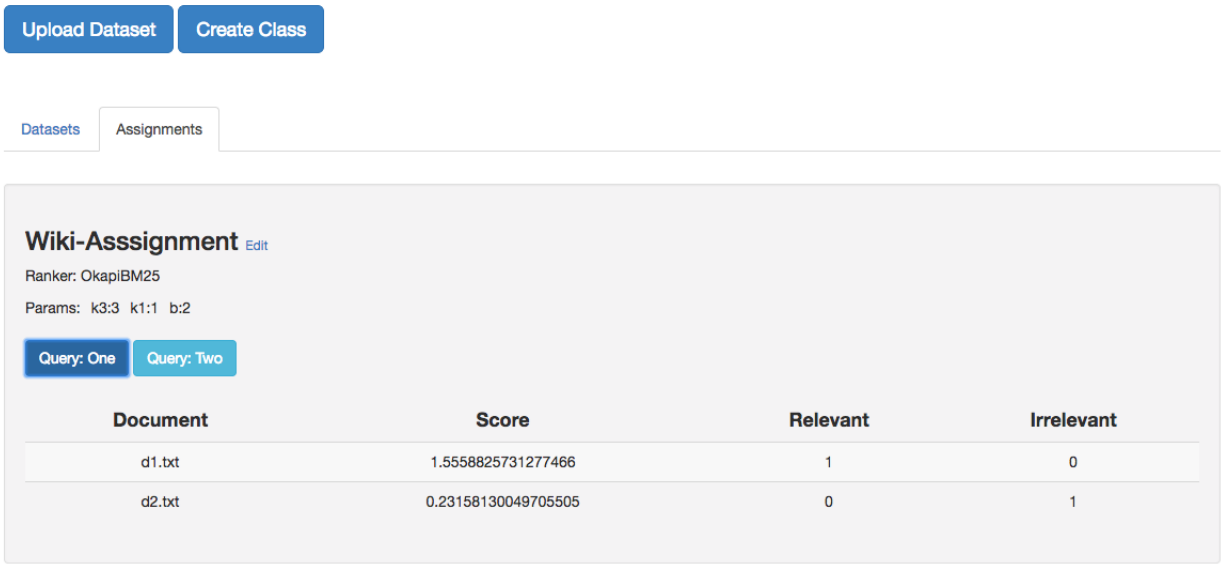

Figure 6.4: Example View Annotation Page

Instructors can view the status of annotations in their interfaces too. As shown in Figure 6.4, instructors can see the number of relevance judgments for relevant and irrelevant respectively. For example, in the interface, for the query "One", there is one judgment stating that "d1" is relevant to the query and one judgment stating that "d2" is irrelevant to the query. Moreover, they can also see the scores from the Search Engine module given by the algorithm and parameters they choose when they create the assignments. It is very useful as it provides instructors an idea of how relevant the documents are to the query from the search engine's perspective.

6.2 ANNOTATOR PERSPECTIVE

In this section, we will show how annotators can use the system and the Search Engine module to complete annotation assignments as well as experiment and explore with different search algorithms.

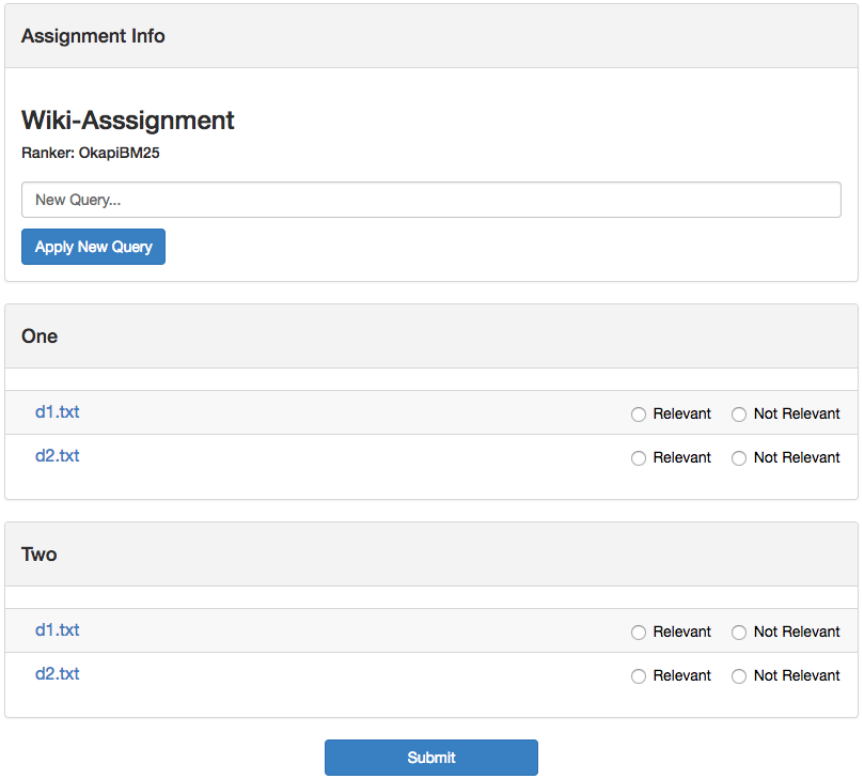

Figure 6.5: Example Annotation Page

When annotators click into one of their annotation assignments, they will see an annotation page as illustrated in Figure 6.5. They can see a list of queries and a number of ranked documents corresponding to the query. For example, the interface shows two queries "One " and "Two" with two lists of documents with relevant or not relevant radio buttons. Annotators can click into the document to see the contents and make annotations accordingly since the Search Engine module can retrieve the contents.

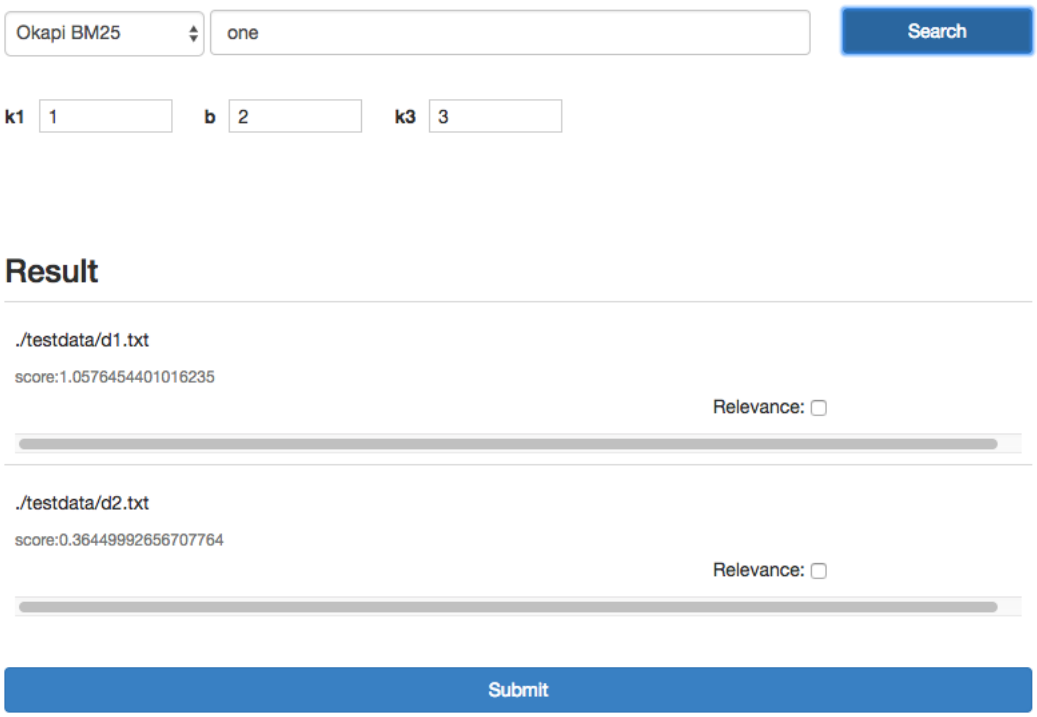

Figure 6.6: Example Search Page

Annotators can also experiment with different algorithms, parameters and queries and provide annotations using the search page powered by the Search Engine module as illustrated in Figure 6.6. In this example, the user has selected "Okapi BM25" as the algorithm, "one" as the query and parameters like "k1", "b" and "k3".

In this chapter we have shown a simple evaluation of how users can use this Annotation Subsystem especially the Search Engine module to publish data set, create annotation assignments and make annotations using real information retrieval algorithms effortlessly.

CHAPTER 7: CONCLUSIONS AND FUTURE WORK

The growth of big data created tons of opportunities for scientists and engineers to exploit many data mining, information retrieval and machine learning techniques to gain actionable knowledge from the data, which can improve productivity and life quality. With so many new opportunities, there is an increasing demand to train and educate a large number of data scientists. Online education platforms such as Coursera can provide Massive Open Online Courses(MOOCs) are effective to educate engineers and scientists to learn data science skills. However, those existing platforms lack the ability of deploying programming assignments which are essential part of data science education.

This thesis presents the design and implementation of an important module (i.e., the Search Engine Module) in the Annotation Subsystem of a novel Cloud-based Open Lab for Data Science (COLDS). COLDS is designed to support data science programming assignment for which the big data sits on the cloud. It can make online education more practical by enabling instructors to deploy programming assignments that need access to big data at a large scale. The Annotation Subsystem is one important component in COLDS where instructors can collaboratively create new data sets using the help from students. Potentially, COLDS will rely on the annotation system to create new data sets and collect annotations for the instructors.

The focus of this thesis is the Search Engine module, which is also a core part of the whole annotation system and provides the infrastructure for instructors to customize new data sets according to their needs. It also provides the ability for all users to experiment with different information retrieval algorithms efficiently by minimizing users' effort of implementing and testing algorithms. The Search Engine addresses many problems and challenges of allowing users to experiment with large data sets which they cannot easily do so in their local environments. The Search Engine module also addressed the challenge of providing a simple but general interface for users to use it for different purposes. The design and implementation of the Search Engine is mainly focusing on making the module scalable, robust and general so that it can address all the challenges. For example, the storage design and implementation makes the Search Engine module more extensible and flexible for many large data sets.

The Search Engine module is also very extensible and general so that there can be many more features added to the entire Annotation Subsystem. From the scalability perspective, the

general design and implementation of the Search Engine module makes it easy to partition and replicate data sets in order to accommodate the rapid growth of big data. In addition, the general interface provided by the Search Engine module enables easy comparisons of performance between different algorithms on some meaningful data sets.

The current Annotation Subsystem can already be deployed to enable instructors of information retrieval courses to design and deploy annotation assignments. We plan to test the system using CS410 Text Information Systems in Fall 2018.
The current Annotation Subsystem is only a basic version of our long-term plan. It can be further extended by including an intelligent distribution of annotation assignments to annotators. For instance, it can be easy for annotators to reach consensus on some query-document pairs and thus easy to get enough annotations for that. An intelligent algorithm may be implemented to avoid showing those pairs repeatedly so that there can be fewer redundant annotations. Moreover, it is also possible to include more search engine implementations into the system so that students can even compare the results between algorithms and some known search engines such as Bing powered by Microsoft. At last, it is essentially important to deploy the whole system on some cloud platforms like AWS, GCP and Azure to exploit computing power there that can suit the needs for both instructors and students in data science.

REFERENCES

[1] G. Li, "Design and implementation of the instructor module and annotator module in colds," M.S. thesis, University of Illinois at Urbana-Champaign, 2017.

[2] C. Liu, "Design and implementation of the database module in colds for data annotation," M.S. thesis, University of Illinois at Urbana-Champaign, 2017.

[3] X. Wei, "Design and implementation of the annotation module in colds," M.S. thesis, University of Illinois at Urbana-Champaign, 2017.

[4] T. Kalil and F. Zhao, "Unleashing the power of big data," 2013. [Online]. Available: https://obamawhitehouse.archives.gov/blog/2013/04/18/unleashing-power-big-data

[5] P. Liang and J. Abernathy, "Mlcomp," 2013. [Online]. Available: http://mlcomp.org/

[6] H. Fang, H. Wu, P. Yang, and C. Zhai, "Virlab: A web-based virtual lab for learning and studying information retrieval models," in *SIGIR 2014 - Proceedings of the 37th International ACM SIGIR Conference on Research and Development in Information Retrieval*. Association for Computing Machinery, 2014, pp. 1249–1250.

[7] J. Howe, "The rise of crowdsourcing," *Wired magazine*, vol. 14, no. 6, pp. 1–4, 2006.

[8] H. Su, J. Deng, and L. Fei-Fei, "Crowdsourcing annotations for visual object detection," in *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*, vol. 1, no. 2, 2012.

[9] A. Wang, C. D. V. Hoang, and M.-Y. Kan, "Perspectives on crowdsourcing annotations for natural language processing," 2010.

[10] K. S. Jones, *Readings in information retrieval*. Morgan Kaufmann, 1997.

[11] M. Sanderson et al., "Test collection based evaluation of information retrieval systems," *Foundations and Trends® in Information Retrieval*, vol. 4, no. 4, pp. 247–375, 2010.

[12] E. M. Voorhees, D. K. Harman et al., *TREC: Experiment and evaluation in information retrieval*. MIT press Cambridge, 2005, vol. 1.

[13] A. Kittur, E. H. Chi, and B. Suh, "Crowdsourcing user studies with mechanical turk," in *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 2008, pp. 453–456.

[14] O. Alonso, D. E. Rose, and B. Stewart, "Crowdsourcing for relevance evaluation," in *ACM SigIR Forum*, vol. 42, no. 2. ACM, 2008, pp. 9–15.

[15] C. Manning, M. Surdeanu, J. Bauer, J. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[16] D. Khashabi, M. Sammons, B. Zhou, T. Redman, C. Christodoulopoulos, V. Srikumar, N. Rizzolo, L. Ratinov, G. Luo, Q. Do et al., "Cogcompnlp: Your swiss army knife for nlp."

[17] P. Ogilvie and J. P. Callan, "Experiments using the lemur toolkit." in *TREC*, vol. 10, 2001, pp. 103–108.

[18] S. Massung, C. Geigle, and C. Zhai, "MeTA: A Unified Toolkit for Text Retrieval and Analysis," in *Proceedings of ACL-2016 System Demonstrations*. Berlin, Germany: Association for Computational Linguistics, August 2016. [Online]. Available: http://anthology.aclweb.org/P16-4016 pp. 91–96.

[19] C. Zhai and J. Lafferty, "A study of smoothing methods for language models applied to ad hoc information retrieval," in *ACM SIGIR Forum*, vol. 51, no. 2. ACM, 2017, pp. 268–276.