NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

# INFORMATION RETRIEVAL WITH CONCEPT ONTOLOGY FOR DOMAIN-SPECIFIC TEXT

LI HAIHUI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
AY 2016/2017

NANYANG TECHNOLOGICAL UNIVERSITY

SCE16-0085

# INFORMATION RETRIEVAL WITH CONCEPT ONTOLOGY FOR DOMAIN-SPECIFIC TEXT

Submitted in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Engineering (Computer Science)
of the Nanyang Technological University

by

**LI HAIHUI**

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
AY 2016/2017

# Abstract

In an age of information boom, efficient retrieval of information is becoming more important. Enormous amount of information can sometimes cause overload for people. Moreover, traditional search engines only return users with a ranked list of documents without a grand overview of information. To address users' growing information needs, we proposed an information retrieval solution with the use of concept ontology. By integrating information retrieval with ontology, users can effectively navigate among different documents and have a quick grasp of the information contained in the documents.

A proof-of-concept web application, named *DSPLearn*, was developed in the domain of digital signal processing. It integrates traditional keyword search with the idea of concept ontology. The technologies behind DSPLearn are generic and can be applied to any kind of text and any other knowledge bases.

DSPLearn supports efficient search of PDF documents. It generates a concept tree based on the search results for a query, from which users can filter the results. It also allows highlighting of terms that are mapped to some user-selected concepts in a PDF document. An $n$-th match approach was proposed to locate an exact term in a document.

With rapid information growth, the idea of concept ontology is promising. Ontology will play a significant part in building a Semantic Web - a Web of data.

# Acknowledgments

This Final Year Project (FYP) was made possible thanks to the cooperation and support of the following people, who have helped me gain much more than what was expected. I am grateful and thankful to them.

First of all, I would like to express my deepest gratitude to Assoc Prof Chng Eng Siong, my FYP supervisor, for his valuable advice and guidance throughout the project.

I also wish to thank Mr. Kyaw Zin Tun, a project officer from Rolls-Royce@NTU Corporate Lab, for his feedback and assistance during software development.

Finally, I wish to express my appreciation to the two undergraduate students, Sun Ximeng and Wu Zhenghao, for their work on the text retrieval project. The web application developed would not be complete without the work they have done.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Background

We are living in an era of information explosion. IBM reported that every day, 2.5 quintillion bytes of data are created – so much that 90% of the data in the world today have been created in the last two years alone [1]. As the amount of available data grows, the problem of managing the information becomes more difficult, which can lead to information overload.

Information overload occurs when individuals are exposed to more information than they can effectively utilize, which causes frustration and anxiety and leads to poor decision making [2]. According to a report from McKinsey, this is a serious issue faced by all types of upper-level management in the modern information economy [3]. With so much overwhelming information available online, we often need uninterrupted time to synthesize information from many different sources to satisfy our needs.

Traditional search engines search documents for specified keywords and returns a list of the documents where the keywords are found, in the order of relevance. Very often, users wish to obtain a brief overview on the long list of results. However, current search engines do not summarize the search results for a query. Most of them neither create linkage among similar documents.

To meet users' growing information needs in today's data boom, we proposed an effective information retrieval solution with the use of concept ontology. A concept ontology, or ontology, is a specification of a shared vocabulary that can be used to model a domain of discourse. Ontology captures definitions and interrelationships of the entities that fundamentally exist for a particular domain [4]. These entities are called *concepts* for that particular domain.

By integrating information retrieval with concept ontology, users can filter search results for a query based on concepts and navigate among different documents effectively and efficiently. For this project, we developed a proof-of-concept information retrieval system in the domain of Digital Signal Processing (DSP).

## 1.2 Aims and Objectives

The aim of this project is to develop a proof-of-concept web application for search in the domain of digital signal processing with the use of concept ontology. This web application would serve as a framework for information retrieval with ontology so that it is readily usable for any other domains of discourse. The objectives for this project are:

1. To develop a web application that supports efficient search of PDF documents;
2. To generate a concept tree based on the search results for a user query, from which users can filter the documents returned;
3. To highlight terms that are mapped to user-selected concepts in a PDF document.

## 1.3 Scope

The domain for information retrieval is restricted to digital signal processing. However, it is possible to use the web application for other domains.

The information retrieval system developed in this project is not a full-fledged product because ontology developing and concept recognition are not within the scope. However, the format for concept recognition output, i.e. term-to-concept mappings, is defined. A simplified ontology is also manually crafted for demonstration purpose.

## 1.4 Report Organization

This thesis is divided into six chapters and an overview of each chapter is as follows:

- Chapter 1 describes background, aims and objectives, and scope for this project;

- Chapter 2 reviews some past literature on ontology, keyword search, and concept search and studies an ontology-based search tool for biomedical research abstracts;
- Chapter 3 presents the methodology for keyword search with ontology, architecture of the proposed information retrieval system, along with the key technology solutions used;
- Chapter 4 describes the environment setup for application development and highlights the key implementation for crawling, indexing, searching, and viewing;
- Chapter 5 demonstrates the functionalities of the web application developed for this project with the aid of screenshots;
- Chapter 6 concludes the thesis and discusses future work that can be done on this topic.

# Chapter 2

# Literature Review

## 2.1 Concept Ontology

Ontologies have received more and more popularity in the area of knowledge management and knowledge sharing, especially after the evolution of the Semantic Web and its supporting technologies [5]. Gruber [6] defined ontology as “a formal specification of a shared conceptualization”. This specification defines the concepts and the relationships among them, which are relevant for modeling a domain.

Ontologies are becoming the cornerstone of the Semantic Web. Ontologies aim at capturing domain knowledge in a generic way and provide a commonly agreed understanding of a domain [4]. The development of ontologies demands the use of various software tools called Ontology Editors. These tools [7] like Protégé, SWOOP, and Neologism can be applied to different stages of the ontology life cycle including creation, implementation, and maintenance of ontologies. In [4], it mentions that developing ontology is an iterative process. The ontology is kept revised and refined from the rough first version after discussing with experts in the field.

To construct an ontology one must have an ontology specification language. The Web Ontology Language (OWL) is the widely accepted Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations among things [8]. Knowledge expressed in OWL can be exploited by computer programs.

## 2.2 Case Study: GoPubMed

GoPudMed is a knowledge-based search engine for 14 million biomedical research abstracts in PubMed [9], with the use of Gene Ontology (GO). According to Delfs et al. [9], it utilizes ontology-based keyword search and structures search results through categories of GO. Once GoPudMed receives user query, it submits the keywords to PubMed, extracts GO-terms from the retrieved abstracts, and presents the relevant sub-ontology for browsing.

The GoPubMed approach for literature search has a few advantages, as noted in [9]. First, instead of pursuing only top hits, users are able to gain a high-level overview of the search results from the induced ontology. Second, the hierarchically ordered ontology allows for fast navigation of the search results from general to specific concepts. Third, the use of Gene Ontology enables one to derive general keywords relevant to the search although they may not be mentioned in the articles at all, as they are derived indirectly from the GO.

## 2.3 Keyword Search vs Concept Search

Information retrieval is concerned with selecting some documents from a collection based on a user's information needs expressed using a query.

In the keyword-based approach, both documents and queries are represented as bags of lexical entities, namely keywords. A keyword can be a simple word or a compound word. As noted in [10], weights are associated with keywords to express their importance in the document (or the query). The weighting scheme is generally based on variations of the well-known *tf*idf* formula.

In the concept-based approach, concepts are identified from the content of the document (or the query), and weighted according to both their frequency distribution and their semantic similarity to other concepts in the document (or the query) [11].

# Chapter 3

# Proposed Methodology and System Specifications

## 3.1 Document Server

### 3.1.1 Introduction

In this thesis, a *document* is defined as a section of a book, which may be contained within other sections of the same book. Each section has a title and may or may not have parent or child sections. Document Server refers to the server set up for storing and retrieving of PDF, processed text, and meta-data of these sections. The meta-data of a section includes the section title, the book that the section is in, the start page from which the section starts, its previous and next sections, its parent section, and whether it has a child section. The Document Server allows users to:

- Store data extracted from PDF e-books in a relational database, including text and meta-data of each section;
- Retrieve a specified version of processed text for a section;
- Post a new version of processed text to the relational database;
- Retrieve meta-data of a section;
- Retrieve a PDF file by specifying start page and end page.

### 3.1.2 Text Extraction from PDF

As mentioned, documents for the proposed information retrieval system are essentially sections of e-books. Thus, we need a way to identify each section of an e-book and then extract section text.

The table of contents of each e-book can serve to identify all sections. The bookmarks in PDF are corrected based on table of contents. These bookmarks are used as references to extract text for each section.

Adobe Acrobat DC was used to correct bookmarks in PDF and convert PDF into HTML. The PDF text is converted into HTML so that the text can be further preprocessed and indexed. The extracted text and meta-data are stored into the relational database.

Adobe Acrobat DC is packed with the tools that enable users to create, edit, print, manage, comment, review, and sign PDF documents in a reliable, consistent manner. Adobe Acrobat DC 15.0 was used for the file operations above.

### 3.1.3 RESTful API for Versioning Data

A RESTful API is an Application Program Interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data. It is based on Representational State Transfer (REST) technology, an approach to communications often used in web services development.

An application[1] has been developed to allow users to manage and maintain a huge database of text extracted from PDF e-books and to access and upload data to the Document Server through RESTful API. This enables easy sharing of text data and boosts collaboration among researchers in a team or across teams.

For example, users may access processed text of a particular version via GET request, further process it, and then upload to the Document Server with a new version via POST request. Users may also read a PDF file by specifying start page and end page via GET request.

The available API for document data are listed in Appendix A. For details on the source code, readers may access the GitHub repository `https://github.com/nathanielove/pdf-server`.

A Python client library has also been developed to provide a more pleasant experience to access the RESTful API. For more details, readers may check out the GitHub repository `https://github.com/nathanielove/pdf-client`.

[1]This application had been developed by the undergraduate student Sun Ximeng when the author started the project.

## 3.2 Normalization

Normalization is the process of canonicalizing tokens so that matches occur despite superficial differences in the character sequences of the tokens [12]. Usual normalization techniques include case folding, lemmatization, stop-word removal, stemming, and so on.

### 3.2.1 Formula Removal

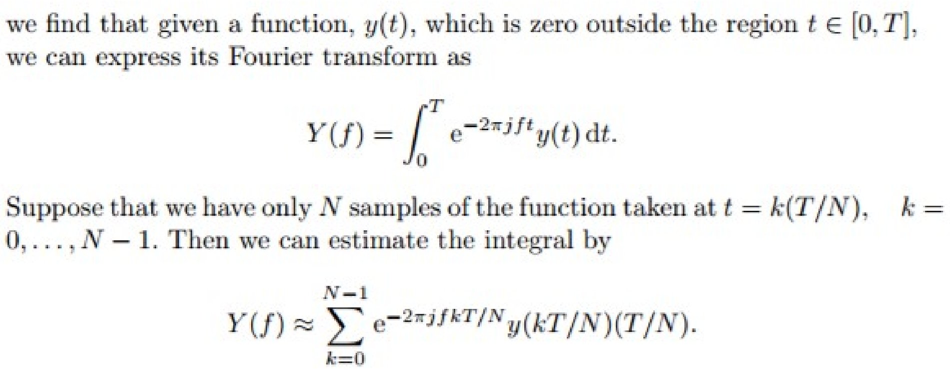
For a technical domain like DSP, there can be many formulas in the text. Formula is a mathematical relationship or rule expressed in symbols. As users would not search for documents based on formulas, those formulas should be removed before indexing documents.

Figure 3.1 demonstrates an example of original PDF text, converted raw text, and the text after removing formulas. As seen, after converting to raw text from PDF, the formulas become human-unreadable and have no useful meaning. A program[2] was thus developed to detect such formulas and replace them with the placeholder `[FORMULA]`.

Dictionary lookup is the approach used to detect formulas and replace them with a placeholder. As shown in Figure 3.2, a document is first tokenized. Each token is an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing [12]. Token preprocessing involves converting a token from upper case to lower case, reducing it to base form, and replacing hyphen with space. The dictionary is a text file with over 354,000 words. A token is flagged as “formula” if it is neither a number nor a word found in the dictionary after preprocessing. Consecutive placeholders or placeholders separated with punctuations are then merged into one by the program developed. For details on the source code, readers may access the GitHub repository `https://github.com/axawzh/TextCleaner`.

The other approach, using regular expressions to detect formulas, was abandoned because it is difficult to maintain a complicated set of regular expressions that can map to a universal set of formulas in a domain. The approach using regular expressions tends to be error-prone and not extensive.

[2]This program was developed by the undergraduate student Wu Zhenghao.



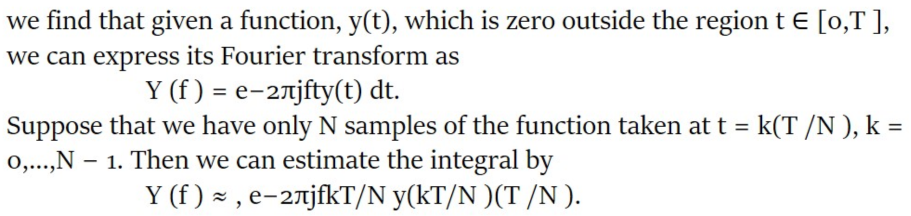
we find that given a function, $y(t)$, which is zero outside the region $t \in [0, T]$, we can express its Fourier transform as

$$Y(f) = \int_0^T e^{-2\pi jft} y(t)\, dt.$$

Suppose that we have only $N$ samples of the function taken at $t = k(T/N)$, $\quad k = 0, \ldots, N-1$. Then we can estimate the integral by

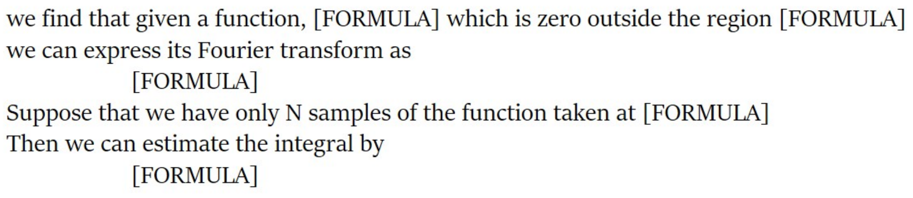
$$Y(f) \approx \sum_{k=0}^{N-1} e^{-2\pi jfkT/N} y(kT/N)(T/N).$$

(a) Sample PDF



we find that given a function, y(t), which is zero outside the region t ∈ [0,T ],
we can express its Fourier transform as
Y (f ) = e−2πjfty(t) dt.
Suppose that we have only N samples of the function taken at t = k(T /N ), k =
0,...,N − 1. Then we can estimate the integral by
Y (f ) ≈ , e−2πjfkT/N y(kT/N )(T /N ).

(b) Sample Raw Text



we find that given a function, [FORMULA] which is zero outside the region [FORMULA]
we can express its Fourier transform as
[FORMULA]
Suppose that we have only N samples of the function taken at [FORMULA]
Then we can estimate the integral by
[FORMULA]

(c) Sample Text after Formula Removal

Figure 3.1: Formula Removal Example

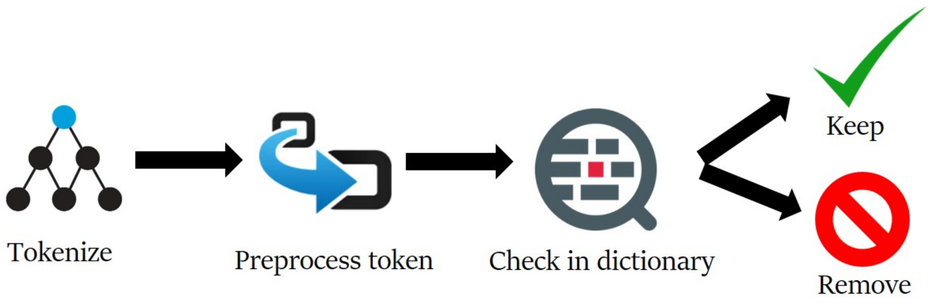


Figure 3.2: Dictionary Lookup

### 3.2.2 Solr Filters

Apache Solr is an open-source Java search engine used for this project. It has built in with some filters for traditional normalization techniques, including stop-word removal, case folding, and stemming.

`schema.xml` is a XML configuration file for Solr. It contains all of the details about which fields documents can contain, and what filters to apply when adding documents to the index, or when querying those fields.

In addition to formula removal, mentioned in Section 3.2.1, there are 5 filters applied to documents during normalization - *Stop Filter*, *Lower Case Filter*, *English Possessive Filter*, *Keyword Marker Filter*, and *Porter Stem Filter*. An additional filter - *Synonym Filter* is applied to query. The functions of each filter are described in Table 3.1.

Table 3.1: Solr Filters

| **Filter** | **Description** |
|---|---|
| Stop Filter | Discards or stops analysis of tokens that are on the given stop words list, named `stopwords.txt`. |
| Lower Case Filter | Converts any upper-case letters in a token to the equivalent lower-case token. |
| English Possessive Filter | Removes possessives (e.g. trailing 's) from words. |
| Keyword Marker Filter | Protects words in the protected word list from being modified by stemmers. |
| Porter Stem Filter | Applies the Porter Stemming Algorithm for English. |
| Synonym Filter | Looks up the token in the list of synonyms and if a match is found, the synonym is emitted in place of the token. |

All these filters are applied before Solr does document indexing. They are configured in `schema.xml`, which can be found in Solr's installation directory.

## 3.3 Term-to-Concept Mappings

Although design of concept recognition algorithm is not within the scope for this project, the output of concept recognition, i.e. term-to-concept mappings, is defined. Each mapping is a tuple - (*Concept*, *Term*, *Section*, *N-th match*). Each item in the tuple is explained below:

- **Concept:** A unique concept node in the ontology tree;
- **Section:** A document, namely a section of an e-book;
- **Term:** a word or a phrase in the Section;

- ***N*-th match:** The $n$-th occurrence of the Term in the Section.

The purpose of adopting $n$-th match approach is to locate the exact term that is mapped to a particular concept in a PDF document. This is because sometimes same terms but in different locations may be mapped to different concepts. For example, suppose the term “bank” appears multiple times in a document. Some occurrences may be mapped to the concept “river side”, while the other may be mapped to the concept “financial organization”.

A term in a document may be mapped to multiple concepts; inversely, a concept may also be mapped to multiple terms.

## 3.4 System Architecture

A web application was developed to demonstrate the use of keyword search with ontology in an information retrieval (IR) system. This IR application consists of two components - one for crawling and indexing and the other for searching and viewing.

### 3.4.1 Crawling and Indexing

As defined by Manning et al. [12], Web crawling is the process by which we gather pages from the Web, in order to index them and support a search engine. In our context, more specifically, it means the process of gathering document data from the Document Server. The crawled text can then be fed into the search engine (Solr in our case) for indexing. Indexing is the process of constructing an inverted index.

Figure 3.3 shows the component architecture for crawling and indexing. During the crawling step, the cleaned text and meta-data of each section would be fetched and stored into MySQL database. The cleaned text has all formulas, including numeric expressions, replaced with the placeholder `[FORMULA]`. The PDF documents for each section are also fetched and stored into the `Media` folder in the Application Server. As for the indexing step, the cleaned text and some meta-data stored in MySQL database would then be fed into Solr for indexing. Additional normalization, mentioned in Section 3.2.2, is done before indexing in Solr.

By right, term-to-concept mappings should have also been fetched and stored into MySQL database during crawling. However, since the concept recognition module has not been developed and is not part of the project

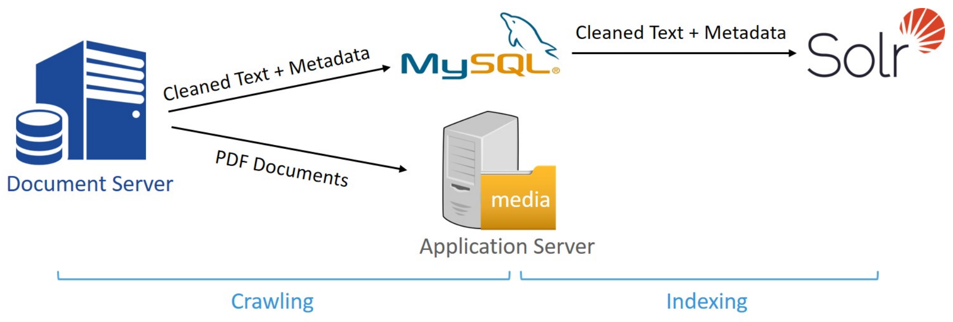


Figure 3.3: Component for Crawling and Indexing

scope, we had to manually fill some mappings data into MySQL database for demonstration purpose. The hierarchical concept ontology was also manually created in the same database.

## 3.4.2 Searching and Viewing

Figure 3.4 shows the component architecture for searching and viewing. The interaction of different components for searching and viewing is elaborated below.

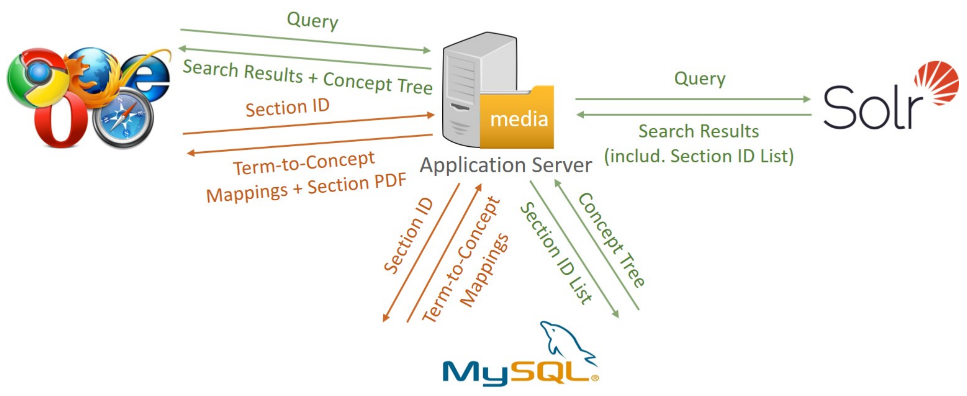

Query
Search Results + Concept Tree
Section ID
Term-to-Concept Mappings + Section PDF
media
Application Server
Query
Search Results (includ. Section ID List)
Section ID
Term-to-Concept Mappings
Section ID List
Concept Tree

Figure 3.4: Component for Searching and Viewing

- **Searching:** Users send a query to the Application Server. The Application Server passes the query to the search engine Solr and then receives the search results from Solr. The search results are a list of documents (i.e. sections) with meta-data including section ID, book, and page number. Next, the Application Server requests for all concepts identified in search results from MySQL. Finally, the Application Server responds to users by sending back the search results and the concept tree. The concept tree consists of concept nodes which map to one or more terms in the documents from the search results.

- **Viewing:** Users request to view one document from the search results. The Application Server receives the section ID. It then retrieves the PDF from the `Media` folder and the term-to-concept mappings for this section from MySQL. Finally, it responds to users by sending back the term-to-concept mappings and the PDF for that document.

## 3.5 Technology Stack

In this part, we introduce the key technology solutions that are utilized for the information retrieval web application, developed in the domain of DSP. How these technology solutions support the IR application is also elaborated.

### 3.5.1 Django

Django is a free and open-source web framework, written in Python, which follows the Model-View-Template (MVT) architectural pattern [13]. Django's automatically generated admin site allows users with right permissions to create/update/delete users and any other database objects specific to Django apps. A Django app is a group of related functionality used to complete or maintain one aspect of a site [14]. A web application may consist of multiple Django apps, depending on how granular each app is.

Django has been an established web framework since 2005. There are many available app plug-ins for extended features. Haystack and Django-Treebeard, which are discussed below, are the two plug-ins used for this IR application. Django also provides well-organized documentation and example code tagged for every specific release.

The core philosophy of Django is DRY: Don't Repeat Yourself. The framework places a premium on getting the absolute most out of very little code. This means less hours and less code to get new features working for developers.

Django is the web framework used for the IR application.

### 3.5.2 MySQL

MySQL is an open-source relational database management system that uses Structured Query Language (SQL). SQL is the most popular language for adding, accessing and managing content in a database. It is most noted for its quick processing, proven reliability, ease and flexibility of use.

MySQL is the database back-end connected to Django for the IR application. It is used to store the cleaned text crawled from Document Server, the meta-data of each book section, the hierarchical ontology, and the term-to-concept mappings of each section.

### 3.5.3 Celery

Celery is an asynchronous task queue based on distributed message passing [15]. As elaborated in [15], it is focused on real-time operation, but supports scheduling as well. The execution units, called tasks, are executed concurrently on a single or more worker servers using multiprocessing. Tasks can execute asynchronously in the background or synchronously (wait until ready). Celery uses a message broker such as Redis to send and receive messages and runs tasks in the background.

Celery, together with the message broker Redis, is used to run the lengthy crawling task in the background and update users on the crawling progress in near real time.

### 3.5.4 Redis

Redis is an open source and in-memory data structure store that persists on disk [16]. It can be used as a database, cache and message broker. The data model is key-value. Many different types of values are supported other than strings, such as lists, sets, and hashes.

Redis is used in the IR application as a message broker for Celery. In a nut shell, it acts as an intermediary that passes messages between the Celery worker server and the Application Server.

### 3.5.5 Solr

Solr is an open-source enterprise search platform built on Apache Lucene, written in Java. It powers some heavily-trafficked websites like eBay, Bloomberg, and Netflix [17].

Solr runs as a standalone full-text search server. It uses the Lucene Java search library at its core for full-text indexing and search and has REST-like API with JSON response [17].

Solr is the search engine used in the IR application. It works with Django via the Django app HayStack, which is mentioned below.

### 3.5.6 HayStack

Haystack is a third-party Django app that supports modular search. It works directly with Django models [18] and provides a familiar API that allows users to plug in different search back-ends (such as Solr, Whoosh, etc) without having to modify the code [19]. With its extensive API, Haystack supports advanced features such as auto-complete and highlighting (i.e. auto-generate document excerpts).

Haystack is used in the IR application for functionalities related to document indexing and keyword search. It can be thought as a linkage between Django and Solr.

### 3.5.7 Django-Treebeard

Django-Treebeard is a third-party Django app that provides efficient tree implementation for Django 1.7+ [20]. It optimizes non-native tree operations such as adding/getting ancestor/child/sibling nodes and provides easy-to-use API. Django-Treebeard also provides drag-and-drop features for easy modification of tree nodes in the Django admin page.

Django-Treebeard is used in the IR application to build hierarchical ontology with Django models.

### 3.5.8 PDF.js

PDF.js is a general-purpose, web standards-based platform for parsing and rendering PDFs [21]. It is an open-source JavaScript library, first released in 2011 by Mozilla Foundation [22].

PDF.js can work as a part of a website or of a browser. It has been included in Mozilla Firefox since 2012 (Version 15) and has been enabled by default since 2013 (Version 19) [22]. There also exists a Chrome extension.

Understood from the official documentation [21], the PDF.js project contains 3 layers - Core, Display, and Viewer. The Core layer is where a binary PDF is parsed and interpreted. This layer is the foundation for all subsequent layers. The Display layer takes the Core layer and exposes an easier-to-use API to render PDFs and get other information out of a document. This API is what the version number is based on. The Viewer layer is built on the Display layer and handles user interactions in the PDF viewer.

PDF.js is used in the IR application to allow users to view, download, print, zoom, and navigate PDF files. The Viewer layer is customized to support the functionality of highlighting terms mapped to selected concepts.

### 3.5.9 jQuery

jQuery is a JavaScript library that greatly simplifies JavaScript programming. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers [23].

jQuery is used extensively for the front-end development of the IR application.

### 3.5.10 Bootstrap

Bootstrap is a very popular HTML, CSS, and JavaScript framework for developing responsive and mobile first web application. It makes front-end web development faster and easier. It is free and open source. Bootstrap 3 supports the latest versions of Google Chrome, Firefox, Internet Explorer, Safari for Mac, and Microsoft Edge.

Bootstrap is used for designing a mobile-friendly and pleasant-looking web UI of the IR application.

# Chapter 4

# Implementation

## 4.1 Environment Setup

The IR web application was developed on Windows 10. Solr, Redis server, MySQL database were hosted in an Ubuntu (16.04) virtual machine.

The programming languages used for the implementation of the IR web application are: Python (3.5.1), JavaScript (1.7), HTML5, and CSS3.

Table 4.1 lists the versions for the technology stack introduced in Section 3.5. Note that the latest Solr version supported by Haystack (2.5.1) is 4.10.2. Celery 3.1 was the latest version that supports Windows at the time of development.

Table 4.1: Technology Stack and Versions

| **Technology** | **Version** |
|---|---|
| Django | 1.10.4 |
| MySQL | 5.7.16-0ubuntu0.16.04.1 |
| Celery | 3.1.18 |
| Redis | 3.2.6 |
| Solr | 4.10.2 |
| Haystack | 2.5.1 |
| Django-Treebeard | 4.1.0 |
| PDF.js | 1.5.188 |
| jQuery | 3.1.1 |
| Bootstrap | 3.3.7 |

The source code is hosted in `https://github.com/HarveyLeo/myfyp`. Readers may refer to the detailed environment setup procedures in Appendix B.

## 4.2 Crawling

### 4.2.1 Django Models

Django models are the single and definitive source of information about data. They contain the essential fields and behaviors of the data that are stored. Generally, each model maps to a single database table.

There are 4 tables created to store the document data, ontology, and term-to-concept mappings in a MySQL database - `Books`, `Sections`, `Concepts`, and `Concept mappings`. The database, together with the `Media` folder (`myfyp/media/`) in the Application Server, stores all the necessary data for searching.

Django crafts the database schema based on the models defined in the model module (`myfyp/dsp_index/models.py`). The module is shown in Appendix C. Practically, the purpose of crawling in this IR application is to fetch data from the Document Server, add entries into the database tables and add PDF files into the `Media` folder.

### 4.2.2 Document Data

A crawler module (`myfyp/dsp_index/crawler.py`) was built to crawl sections of 5 DSP e-books from the Document Server. The crawler module contains two methods that crawl e-books and sections respectively - `crawl_books()` and `crawl_sections()`.

- `crawl_books()`: Fetch e-book meta-data and store into the `Books` table; and fetch e-book PDF files and store into the `Media` folder.
- `crawl_sections()`: Fetch cleaned text (with formulas removed) and meta-data of all sections and store into the `Sections` table; and fetch PDF files of each section and store into the `Media` folder.

In total, 1000 sections were crawled. The `Books` table has fields `Book id`, `Title`, `Pages`, and `Pdf`. The `Sections` table has fields `Section id`, `Title`, `Text`, `Book`, `Page`, and `Pdf`. The description of each field is in Table 4.2.

Celery is used to run the time-consuming crawling task in the background. The URL of the message broker for Celery is set in the project settings file (`myfyp/myfyp/settings.py`). The Celery task module (`myfyp/dsp_index/tasks.py`) is dependent on the crawler module. A single task, named `crawl_task`, is defined in the task module. The task calls the two crawling methods in the crawler module. This task is triggered when `crawl_task.delay()` is called.

Table 4.2: Field Description for Crawled Document Data

(a) Table `Books` Field Description

| Field | Description |
|---|---|
| `Book id` | ID of the book |
| `Title` | Book title |
| `Pages` | Total number of pages of the book |
| `Pdf` | File path to the book's PDF file inside the `Media` folder |

(b) Table `Sections` Field Description

| Field | Description |
|---|---|
| `Section id` | ID of the section |
| `Title` | Section title |
| `Text` | Section text after formula removal |
| `Book` | A foreign key to the `Book` table |
| `Page` | Start page of the section in the book |
| `Pdf` | File path to the section's PDF file inside the `Media` folder |

### 4.2.3 Ontology and Term-to-Concept Mappings

Two additional tables, `Concepts` and `Concept Mappings`, were created to store the ontology and term-to-concept mappings. The `Concepts` table has fields `Label`, `Name`, `Position`, and `Relative to`. The `Concept Mappings` table has fields `Concept`, `Term`, `Section`, and `Nth match`. The description of each field is in Table 4.3.

As mentioned in Section 3.4.1, both ontology and term-to-concept mappings have to be manually created for demonstration purpose. Appendix D illustrates how to create a simple ontology via Python shell.

## 4.3 Indexing

### 4.3.1 Haystack `SearchIndex`

The Haystack configurations, including which search engine to use, are set in the project settings file (`myfyp/myfyp/settings.py`).

A search index module (`myfyp/dsp_index/search_indexes.py`) was built

Table 4.3: Field Description for Ontology and Concept Mappings

(a) Table `Concepts` Field Description

| Field | Description |
|---|---|
| `Label` | Concept label (e.g. 0, 1, 2, 1.1, 2.1.2, etc.) |
| `Name` | Concept name |
| `Position` | Relation to the concept `Relative to`, either `Child of` or `Sibling of` |
| `Relative to` | A foreign key to the `Concepts` table |

(b) Table `Concept Mappings` Field Description

| Field | Description |
|---|---|
| `Concept` | A foreign key to the `Concepts` table |
| `Term` | A word or a phrase in the section |
| `Section` | A foreign key to the `Sections` table |
| `Nth match` | $N$-th occurrence of the term in the section |

to define search index for each Django model. `SearchIndex` objects are the way Haystack determines what data should be placed in the search index and handles the flow of data into the search engine. Generally, a unique `SearchIndex` is created for each Django model that needs to be indexed.

A class named `SectionIndex` was created to correspond to the `Section` model. It inherits two Haystack built-in classes - `indexes.SearchIndex` and `indexes.Indexable`. Below is the code snippet.

```python
class SectionIndex(indexes.SearchIndex, indexes.Indexable):
    text = indexes.CharField(document=True, use_template=True)
    book = indexes.CharField(model_attr='book', faceted=True)
    page = indexes.CharField(model_attr='page')
    sid = indexes.CharField(model_attr='section_id')
    # Add content_auto for autocomplete
    content_auto = indexes.EdgeNgramField(model_attr='text')

    def get_model(self):
        return Section
```

Every `SearchIndex` requires there be one (and only one) field with `document=True`. This indicates to both Haystack and the search engine about which field is the primary field for searching within.

Additionally, we added `use_template=True` on the `text` field. This allowed us to use a data template to build the document the search engine would index. The data template is located in `myfyp/dsp_index/templates/search/indexes/dsp_index/section_text.txt`. The template has the following 2 lines of code:

```
{{object.title|safe}}
{{object.text}}
```

As seen above, two fields in the `Section` model - `title` and `text` are indexed by the search engine Solr. The two fields correspond to `Title` and `Text` in the `Sections` table. The search results for a query, returned from Solr, is JSON-formated with fields defined in the `SectionIndex` class.

### 4.3.2 Index Building

Haystack ships with a command to make index building easy. Once data are all crawled and stored into MySQL database, the index can be built in Solr by running the command:

```
python manage.py rebuild_index
```

The `rebuild_index` command would put data in the database into the Solr index. The index built can be subsequently updated with the command below:

```
python manage.py update_index
```

## 4.4 Searching

### 4.4.1 Searching and Filtering

A search form module (`myfyp/dsp_search/forms.py`) was built to accept user query, pass the query to Solr, and retrieve search results. The module has a class named `SectionSearchForm`, which is inherited from Haystack's built-in class `SearchForm`.

The search form consists of a single field `q`, which is for query. Upon searching, the form will take the cleaned content of the `q` field. The searching process is done by Haystack via the `search()` method. The method returns a collection of documents, or a *queryset* in Django's terms. Below is the code snippet.

```
def search(self):
    """ Override search() from parent class. """
    to_have = json.loads(self.data.get(self.with_field, '[]'))
    not_to_have = json.loads(self.data.get(self.without_field, '[]'))
    sqs = super(SectionSearchForm, self).search().models(Section)
    to_remove = self.get_removed_sids(sqs, to_have, not_to_have)
    return sqs.exclude(sid__in=to_remove)
```

`to_have` and `not_to_have` are two lists of concept IDs from the ontology. Each document in the search results must contain concepts specified in `to_have` and must not contain concepts specified in `not_to_have`. `to_remove` is a list of section IDs that indicate sections to be removed based on the two conditions enforced.

The functionality of filtering search results is realized by passing `to_have` and `not_to_have` arguments via parameters in GET requests. The `search()` method first retrieves documents that match the query and then excludes those that are in the `to_remove` list. The `to_remove` list is generated based on `to_have` and `not_to_have` conditions.

### 4.4.2 Search Results View

A Django view is a Python function or a class that takes a Web request and returns a Web response. This response can be the HTML content of a Web page, a JSON document, a 404 error, or anything. The view itself contains logic that is necessary to return that response. The convention is to put views in files called `views.py` in Django's application directories.

A class named `SectionSearchView` was built to receive users' query request and return response back to users. `SectionSearchView` is dependent on the form class `SectionSearchForm` discussed earlier.

The `SectionSearchView` class includes the following instance methods:

- `get_results_count(self)`: Get the total number of sections that match the query;
- `get_section_list(self)`: Get the list of IDs of sections that match the query;
- `get_concept_tree(self)`: Get a JSON-formated concept tree that corresponds to the search results;

- `get_section_count(self)`: Get a JSON-formatted dictionary describing the number of sections for each concept in the concept tree;
- `get_context_data(self, *args, **kwargs)`: Get a dictionary representing the template context.

Once receiving the queryset returned from `SectionSearchForm`'s `search()` method, `SectionSearchView` computes the number of search results via the `get_results_count()` method. It also gets the concept tree of search results with `get_concept_tree()` method. More details on concept tree generation are discussed in Section 4.4.3. To indicate the number of sections that contain each concept in the concept tree, `get_section_count()` is then called. Lastly, the view `SectionSearchView` uses a Django template (`myfyp/dsp_search/results_page.html`) and the template context returned from `get_context_data()` to generate HTML response.

### 4.4.3 Concept Tree Generation for Search Results

Concept tree is a hierarchical tree structure that represents concepts related to the search results. In this thesis, *immediate concepts* for a document refer to concepts that can be found in the `Concept mappings` table for the document. On the contrary, *related concepts* for a document are those immediate concepts and their ancestor concepts. Thus, concept tree can also be thought as a set of related concepts for all documents in the search results.

The class `ConceptDictionaryGenerator`, located in `myfyp/dsp_search/views.py`, is used to generate the dictionarized concept tree as well as the section counts for each concept in the concept tree.

The class has an attribute named `section_list`, which is a list of section IDs representing the search results. It has the following two methods called from the `SectionSearchView` class:

- `dictionarize_concept_hierarchy(self)`: Get a dictionarized concept tree based on the sections represented by `section_list`;
- `get_section_counts(self)`: Get a dictionary describing the number of sections for each concept in the concept tree;

In a nut shell, generating concept tree is realized by:

1. Retrieve immediate concepts for each document in the search results.

2. For each immediate concept, find the concept path from the root node to the current concept in the ontology.

3. Merge all concept paths into a concept tree, represented by JSON or a Python dictionary.

Generating section counts for each concept in the concept tree is realized by:

1. Initialize counts for all related concepts to be 0.

2. For each document, retrieve all related concepts to the document and increment the counts for these concepts by 1.

Python dictionary is the main data structure used to represent concept tree and concept path. There are instance methods in the class `ConceptDictionaryGenerator` for specific steps above.

## 4.5 Viewing

### 4.5.1 Section Details View

A class named `SectionDetailsView` was built to receive users' request for viewing a section. The class includes the following instance methods:

- `get_concept_tree(self)`: Get a JSON-formated concept tree that corresponds to the section;

- `get_context_data(self, *args, **kwargs)`: Get a dictionary representing the template context, including `book_id`, `section_id`, and `concept_tree`.

The view `SectionDetailsView` uses a Django template (`myfyp/dsp_search/details_page.html`) and the template context returned from `get_context_data()` to generate HTML response. The template is embedded with a PDF viewer using HTML's `<object>` tag. Below is the relevant code snippet from the template.

```
<object id="pdf-viewer"
        class="well"
        type="text/html"
        data="{% url 'pdf_viewer' book=book_id section=section_id %}">
</object>
```

The `data` attribute above specifies the URL of the resource (i.e. the PDF viewer) to be used by `object`. The context variables `book_id` and `section_id` are passed to the URL so that the section's PDF file can be rendered in the PDF viewer.

### 4.5.2 PDF Viewer

The PDF viewer is a HTML file supported by PDF.js, located in `myfyp/dsp_search/pdf_viewer.html`. It is also a Django template that corresponds to the view class `PDFView`.

The class `PDFView` includes the method `get_context_data()`. This method captures URL named groups specifying the section ID and the book ID; then it generates a URL pointing to the section's PDF file. The URL to the PDF file is then passed to the context variable `pdf_url`. Below is the code snippet in `pdf_viewer.html` for opening a PDF file via URL.

```
<script>
  pdfjsWebLibs.pdfjsWebApp.PDFViewerApplication.open('{{ pdf_url }}');
</script>
```

### 4.5.3 Concept Tree Generation for a Section

The mechanism for generating concept tree for a section is similar to generating for a collection of sections. It uses the same class `ConceptDictionaryGenerator`, mentioned in Section 4.4.3. To generate a concept tree for a section, the attribute `section_list` in the class `ConceptDictionaryGenerator` is set to the ID of the section.

### 4.5.4 Term Highlighting in PDF

#### Retrieving (term, $n$-th match) Pairs

A view function named `concept_to_terms()` was built to respond to users' request for term-to-concept mappings. It is located in `myfyp/dsp_search/views.py`.

Given a section, retrieving term-to-concept mappings for a list of concepts is realized by:

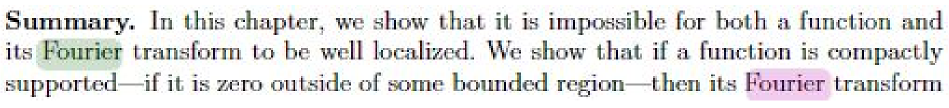
1. Capture the GET parameters from URL. The parameters specify the section ID and the list of concepts.

2. For each concept, look up the `Concept mappings` table and get a list of (term, $n$-th match) pairs.

3. Create a dictionary with keys being the concepts and values being the corresponding lists of (term, $n$-th match).

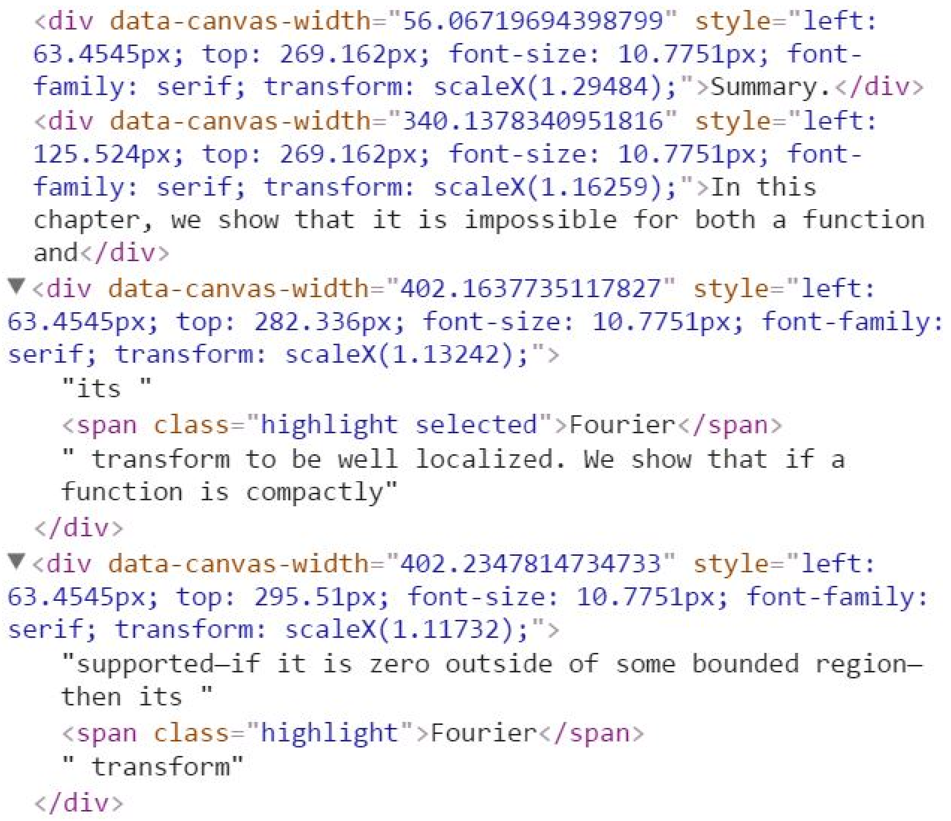
### Highlighting Terms in PDF

The viewer layer of PDF.js is a JavaScript file located in `myfyp/dsp_search/static/dsp_search/js/pdf_viewer.js`. It supports text selection for the rendered PDF by building a text layer on top of non-selectable PDF text. As seen in Figure 4.1, it does this by creating overlaying `div`s over the PDF text. These `div`s contain text that matches the PDF text they are overlaying.



**Summary.** In this chapter, we show that it is impossible for both a function and its Fourier transform to be well localized. We show that if a function is compactly supported—if it is zero outside of some bounded region—then its Fourier transform

(a) Rendered PDF Snippet



```
  <div data-canvas-width="56.06719694398799" style="left:
  63.4545px; top: 269.162px; font-size: 10.7751px; font-
  family: serif; transform: scaleX(1.29484);">Summary.</div>
  <div data-canvas-width="340.1378340951816" style="left:
  125.524px; top: 269.162px; font-size: 10.7751px; font-
  family: serif; transform: scaleX(1.16259);">In this
  chapter, we show that it is impossible for both a function
  and</div>
▼<div data-canvas-width="402.1637735117827" style="left:
63.4545px; top: 282.336px; font-size: 10.7751px; font-family:
serif; transform: scaleX(1.13242);">
    "its "
    <span class="highlight selected">Fourier</span>
    " transform to be well localized. We show that if a
    function is compactly"
  </div>
▼<div data-canvas-width="402.2347814734733" style="left:
63.4545px; top: 295.51px; font-size: 10.7751px; font-family:
serif; transform: scaleX(1.11732);">
    "supported—if it is zero outside of some bounded region—
    then its "
    <span class="highlight">Fourier</span>
    " transform"
  </div>
```

(b) Text Layer of the PDF Snippet

Figure 4.1: Text Layer of Rendered PDF

Figure 4.1 shows a PDF snippet rendered by PDF.js and its corresponding text layer. As seen, each `div` is rendered with a specific position to overlay the underlying layer. The two occurrences of the highlighted term "Fourier" are both enclosed by `<span>` tag and have CSS classes. Both CSS classes `highlight` and `select` have an attribute `background-color` to specify the span's background. Therefore, terms in a PDF document can be highlighted by enclosing each term with `<span>` tag and adding a CSS class to specify the background color.

To locate an exact term in a document given the $n$-th match, jQuery's `find()` function can be used. Below is the code snippet from `myfyp/dsp_search/static/dsp_search/js/details_page.js`. This jQuery statement finds the $n$-th occurrence of a term and enclose it with `<span>` tag, along with some necessary attributes.

```
$pdf_viewer
    .find("#viewer .textLayer > div:icontains('" + term + "')")
    .each(function() {
      var matches = getImatchIndexes($(this).text(), term);

      for (var i = 0; i < matches.length; i++) {
        count++;
        if (nth_match.indexOf(count) > -1) {
          var text = $(this).text(),
              $span = $("<span></span>"),
              cname = $(".tree li a[data-concept-label='" +
                  concept_label + "']").text();
          $span.attr({
            "data-concept-label": concept_label,
            "title": "Concept: " + cname
          });
          if (show_highlight) {
            $span.addClass("highlight mapping");
          }
          $span.text(term);
          $(this).html(text.substr(0, matches[i]) +
              $span[0].outerHTML +
              text.substr(matches[i] + term.length));
        }
      }
    });
```

# Chapter 5

# System Demonstration

## 5.1 Overview

An information retrieval web application in the domain of digital signal processing had been developed. It is called **DSPLearn**. DSPLearn is mobile-friendly and responsive. There are 5 main webpages for DSPLearn - homepage, result page, detail page, dsplearn-admin page, and django-admin page. Below is the description of each page.

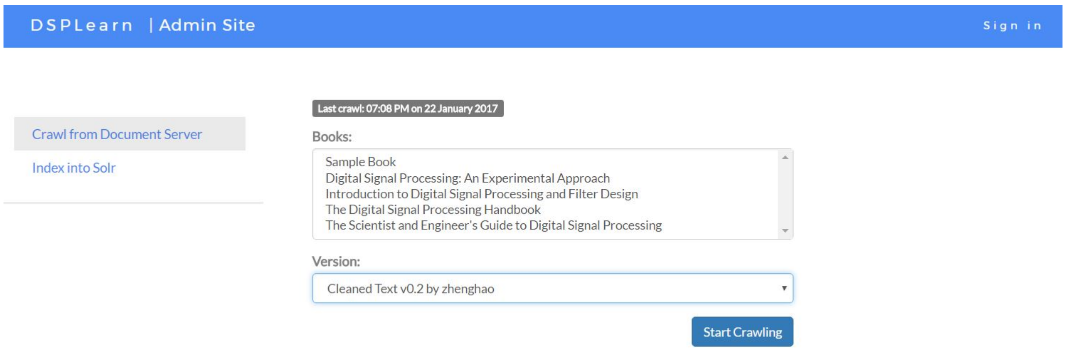
- **Homepage:** The homepage of DSPLearn that features a search bar
- **Result Page:** The page for search results that is navigated to when users do a search
- **Detail Page:** The page to show all details about a section, including the PDF document and related concepts
- **Dsplearn-Admin Page:** The page for crawling document data from the Document Server
- **Django-Admin Page:** The page for viewing and editing document data, ontology, and term-to-concept mappings

## 5.2 Crawling

### 5.2.1 Document Data

The dsplearn-admin page was built to allow admins to crawl document data from the Document Server and store into the MySQL database. There are multiple books and text versions in the Document Server. As shown in Figure

5.1, the page displays the date and time of last crawl and allows selection of books and a text version for crawling.



Last crawl: 07:08 PM on 22 January 2017

Books:

Sample Book
Digital Signal Processing: An Experimental Approach
Introduction to Digital Signal Processing and Filter Design
The Digital Signal Processing Handbook
The Scientist and Engineer's Guide to Digital Signal Processing

Version:

Cleaned Text v0.2 by zhenghao

Figure 5.1: DSPLearn Page for Crawling

The crawling process is run in the background. Users are kept updated of the crawling status in the web interface, as shown in Figure 5.2a.

When some crawling process is running, any other requests for crawling, either from the same user or other users, would be denied, as shown in Figure 5.2b.

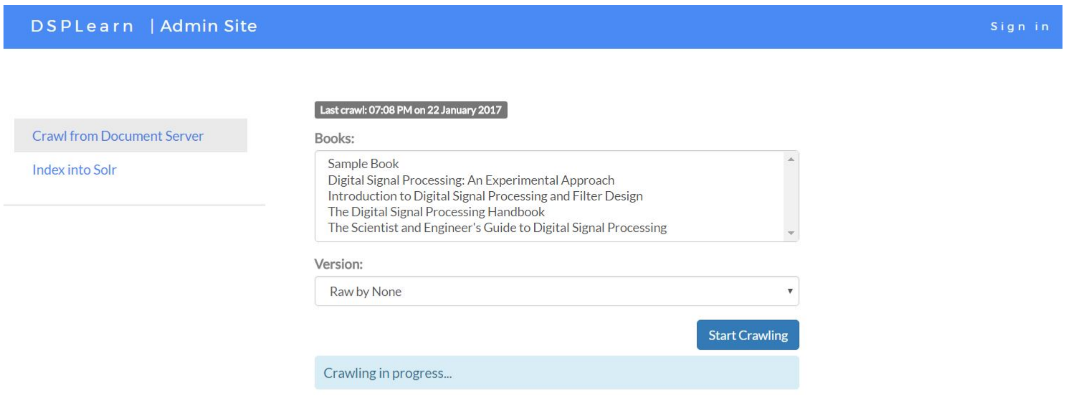
When crawling is successfully done, a success message would appear (Figure 5.2c) and all crawled document data can be accessed in the MySQL database and the `Media` folder.

As seen in Figure 5.3, all PDF files are organized into two directories `books/` and `sections/`, with book or section IDs being the file names.

Figure 5.4 is the django-admin page. You may access data for all 4 table - `Books`, `Concept mappings`, `Concepts`, and `Sections` from this page. All entries in the tables `Books` and `Sections` are filled during crawling (Figure 5.5).

### 5.2.2 Ontology and Term-to-Concept Mappings

The concept ontology and the term-to-concept mappings are both stored in the MySQL database. Users may add new concept into the existing ontology or drag and drop to modify the ontology (Figure 5.6).



Last crawl: 07:08 PM on 22 January 2017

Books:

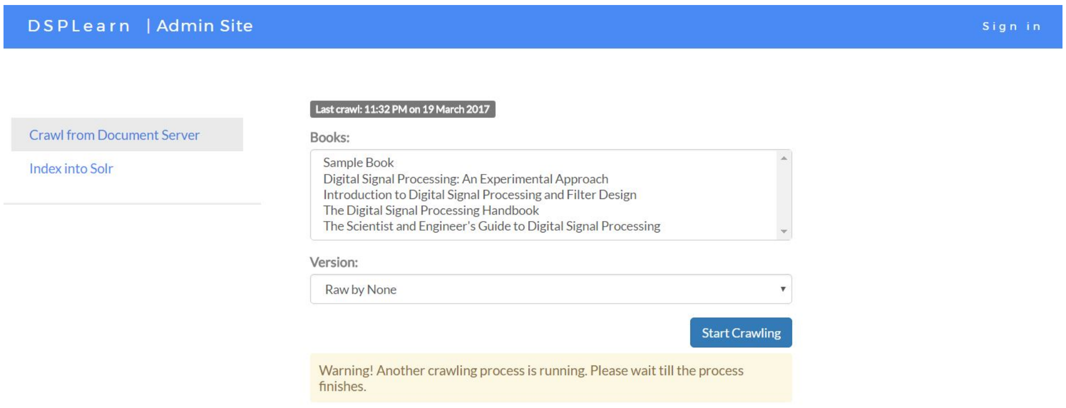
Sample Book
Digital Signal Processing: An Experimental Approach
Introduction to Digital Signal Processing and Filter Design
The Digital Signal Processing Handbook
The Scientist and Engineer's Guide to Digital Signal Processing

Version:

Raw by None

Crawling in progress...

(a) In Progress



Last crawl: 11:32 PM on 19 March 2017

Books:

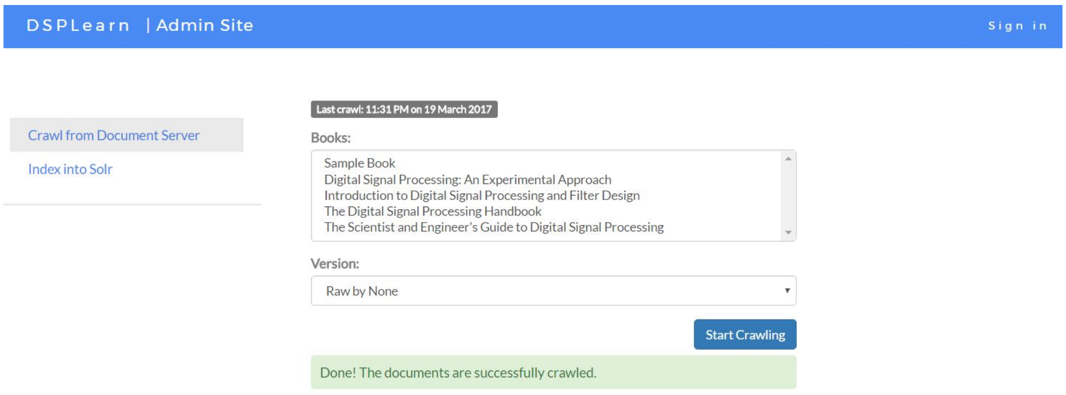
Sample Book
Digital Signal Processing: An Experimental Approach
Introduction to Digital Signal Processing and Filter Design
The Digital Signal Processing Handbook
The Scientist and Engineer's Guide to Digital Signal Processing

Version:

Raw by None

Warning! Another crawling process is running. Please wait till the process finishes.

(b) Another Process Running



Last crawl: 11:31 PM on 19 March 2017

Books:

Sample Book
Digital Signal Processing: An Experimental Approach
Introduction to Digital Signal Processing and Filter Design
The Digital Signal Processing Handbook
The Scientist and Engineer's Guide to Digital Signal Processing

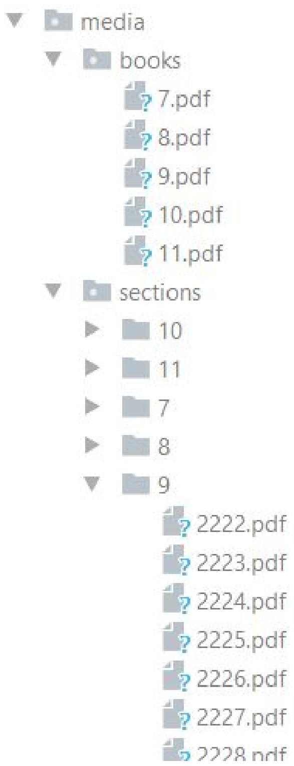
Version:

Raw by None

Done! The documents are successfully crawled.

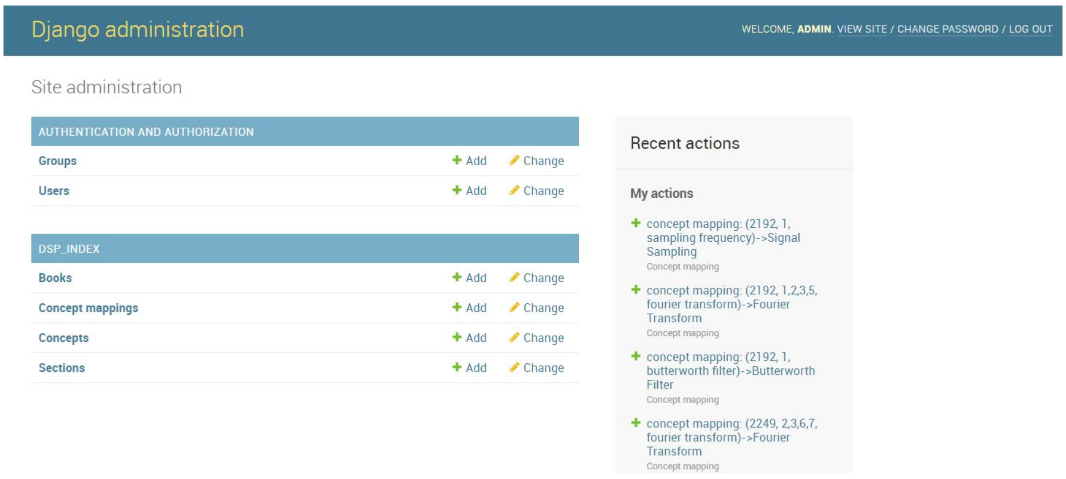
(c) Success

Figure 5.2: Crawling Status Update



```
▼ media
  ▼ books
      7.pdf
      8.pdf
      9.pdf
      10.pdf
      11.pdf
  ▼ sections
    ▶ 10
    ▶ 11
    ▶ 7
    ▶ 8
    ▼ 9
        2222.pdf
        2223.pdf
        2224.pdf
        2225.pdf
        2226.pdf
        2227.pdf
        2228.pdf
```

Figure 5.3: `Media` folder in the Application Server



Django administration

WELCOME, **ADMIN**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

| AUTHENTICATION AND AUTHORIZATION | | |
|---|---|---|
| **Groups** | + Add | Change |
| **Users** | + Add | Change |

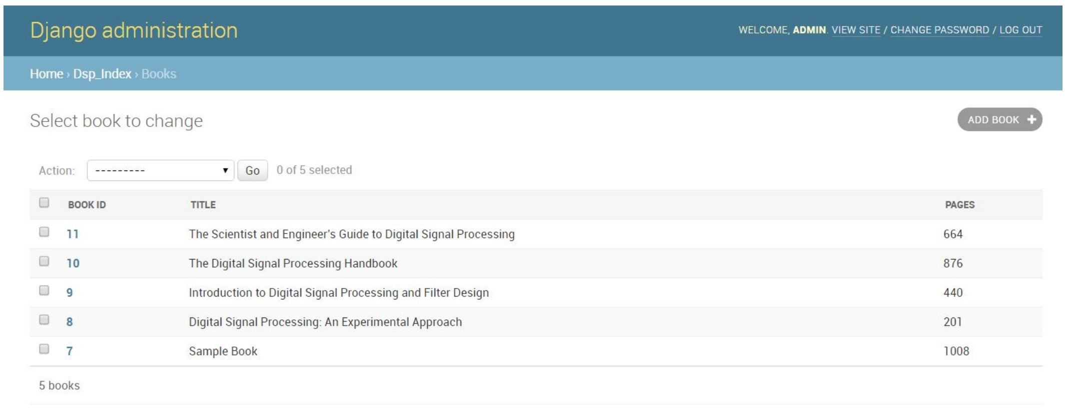
| DSP_INDEX | | |
|---|---|---|
| **Books** | + Add | Change |
| **Concept mappings** | + Add | Change |
| **Concepts** | + Add | Change |
| **Sections** | + Add | Change |

Recent actions

**My actions**

- + concept mapping: (2192, 1, sampling frequency)->Signal Sampling
  Concept mapping
- + concept mapping: (2192, 1,2,3,5, fourier transform)->Fourier Transform
  Concept mapping
- + concept mapping: (2192, 1, butterworth filter)->Butterworth Filter
  Concept mapping
- + concept mapping: (2249, 2,3,6,7, fourier transform)->Fourier Transform
  Concept mapping

Figure 5.4: Django Admin Page


Django administration

WELCOME, **ADMIN**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Dsp_Index › Books

Select book to change

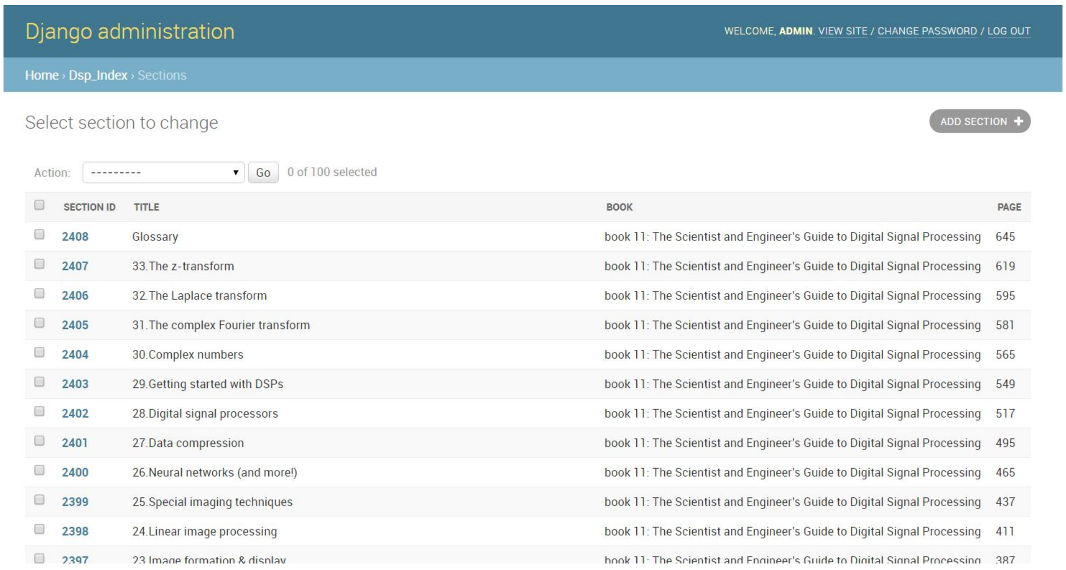
ADD BOOK

Action: --------- Go 0 of 5 selected

| BOOK ID | TITLE | PAGES |
|---|---|---|
| 11 | The Scientist and Engineer's Guide to Digital Signal Processing | 664 |
| 10 | The Digital Signal Processing Handbook | 876 |
| 9 | Introduction to Digital Signal Processing and Filter Design | 440 |
| 8 | Digital Signal Processing: An Experimental Approach | 201 |
| 7 | Sample Book | 1008 |

5 books

(a) Books


Django administration

WELCOME, **ADMIN**. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Dsp_Index › Sections

Select section to change

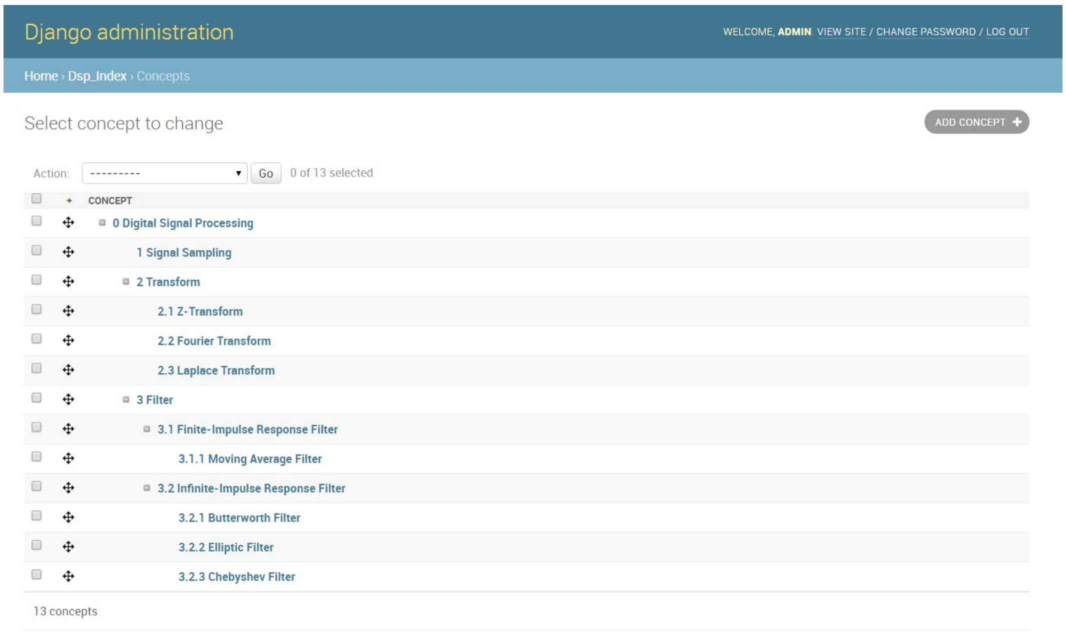
ADD SECTION

Action: --------- Go 0 of 100 selected

| SECTION ID | TITLE | BOOK | PAGE |
|---|---|---|---|
| 2408 | Glossary | book 11: The Scientist and Engineer's Guide to Digital Signal Processing | 645 |
| 2407 | 33.The z-transform | book 11: The Scientist and Engineer's Guide to Digital Signal Processing | 619 |
| 2406 | 32.The Laplace transform | book 11: The Scientist and Engineer's Guide to Digital Signal Processing | 595 |
| 2405 | 31.The complex Fourier transform | book 11: The Scientist and Engineer's Guide to Digital Signal Processing | 581 |
| 2404 | 30.Complex numbers | book 11: The Scientist and Engineer's Guide to Digital Signal Processing | 565 |
| 2403 | 29.Getting started with DSPs | book 11: The Scientist and Engineer's Guide to Digital Signal Processing | 549 |
| 2402 | 28.Digital signal processors | book 11: The Scientist and Engineer's Guide to Digital Signal Processing | 517 |
| 2401 | 27.Data compression | book 11: The Scientist and Engineer's Guide to Digital Signal Processing | 495 |
| 2400 | 26.Neural networks (and more!) | book 11: The Scientist and Engineer's Guide to Digital Signal Processing | 465 |
| 2399 | 25.Special imaging techniques | book 11: The Scientist and Engineer's Guide to Digital Signal Processing | 437 |
| 2398 | 24.Linear image processing | book 11: The Scientist and Engineer's Guide to Digital Signal Processing | 411 |
| 2397 | 23.Image formation & display | book 11: The Scientist and Engineer's Guide to Digital Signal Processing | 387 |

(b) Sections

Figure 5.5: Django Admin Page for `Sections` and `Books`

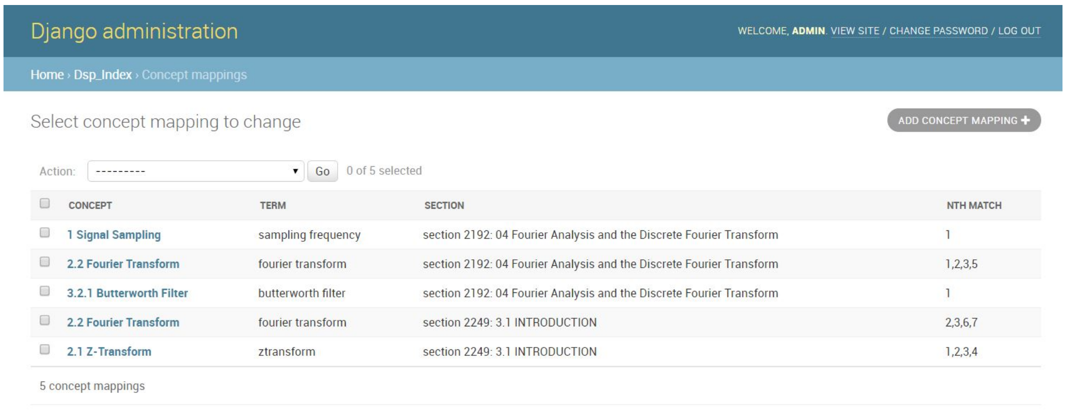

Django administration
WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Dsp_Index › Concepts
ADD CONCEPT
Action: --------- Go
0 of 13 selected

Select concept to change

| CONCEPT |
|---|
| 0 Digital Signal Processing |
| 1 Signal Sampling |
| 2 Transform |
| 2.1 Z-Transform |
| 2.2 Fourier Transform |
| 2.3 Laplace Transform |
| 3 Filter |
| 3.1 Finite-Impulse Response Filter |
| 3.1.1 Moving Average Filter |
| 3.2 Infinite-Impulse Response Filter |
| 3.2.1 Butterworth Filter |
| 3.2.2 Elliptic Filter |
| 3.2.3 Chebyshev Filter |

13 concepts

(a) `Concepts`


Django administration
WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Dsp_Index › Concept mappings
ADD CONCEPT MAPPING
Action: --------- Go
0 of 5 selected

Select concept mapping to change

| CONCEPT | TERM | SECTION | NTH MATCH |
|---|---|---|---|
| 1 Signal Sampling | sampling frequency | section 2192: 04 Fourier Analysis and the Discrete Fourier Transform | 1 |
| 2.2 Fourier Transform | fourier transform | section 2192: 04 Fourier Analysis and the Discrete Fourier Transform | 1,2,3,5 |
| 3.2.1 Butterworth Filter | butterworth filter | section 2192: 04 Fourier Analysis and the Discrete Fourier Transform | 1 |
| 2.2 Fourier Transform | fourier transform | section 2249: 3.1 INTRODUCTION | 2,3,6,7 |
| 2.1 Z-Transform | ztransform | section 2249: 3.1 INTRODUCTION | 1,2,3,4 |

5 concept mappings

(b) `Concept mappings`

Figure 5.6: Django Admin Page for `Concepts` and `Concept Mappings`

## 5.3 Indexing

There is no webpage built for indexing as indexing can be done via a command in the prompt, as mentioned in Section 4.3.2.

## 5.4 Searching

Figure 5.7 shows the homepage for DSPLearn, which features a search bar.

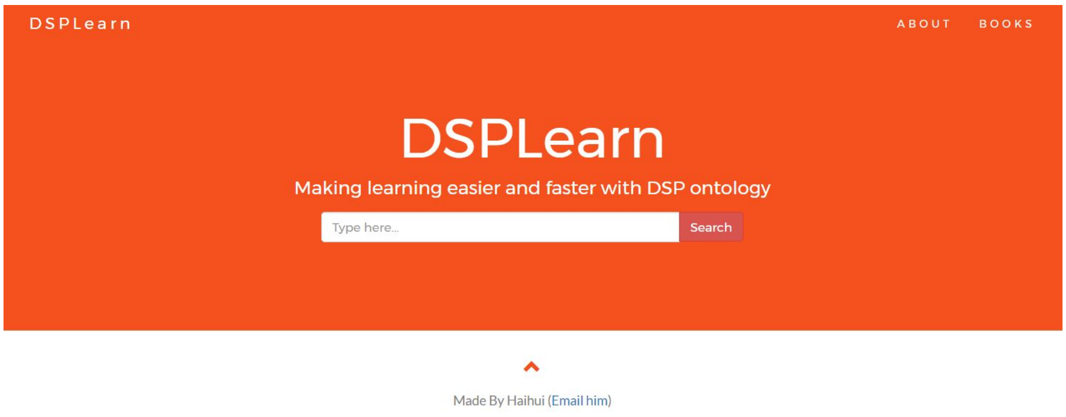
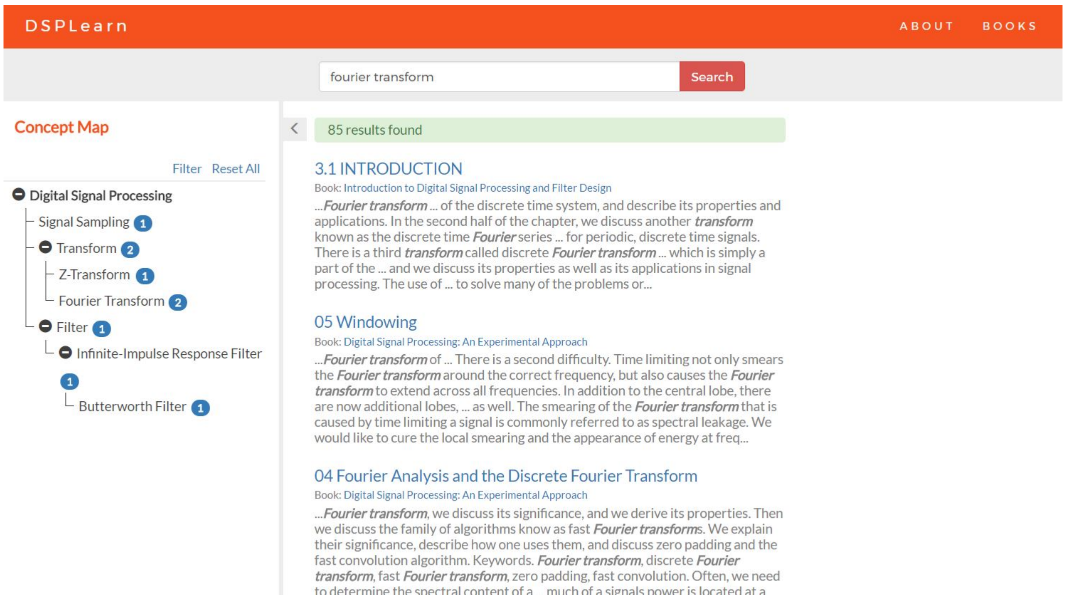

DSPLearn
ABOUT
BOOKS
DSPLearn
Making learning easier and faster with DSP ontology
Type here...
Search
Made By Haihui (Email him)

Figure 5.7: DSPLearn Homepage

When a query is entered, the browser would be directed to the result page (Figure 5.8). As seen, users can also enter new queries in the search bar on top of the result page. The left sidebar features a hierarchical concept tree. The tree consists of concepts that are related to one or more documents in the search results. The blue badge near each concept indicates the number of documents in the search results that are related to the concept. The total number of search results is shown in the green notification bar. The section title, the book the section is in, and the excerpt are displayed for each document in the search results. The excerpt has a maximum of 500 characters and the keywords from the query are in bold. Those . . . placeholders are replacement of `[FORMULA]` from the document text.

The search results are paginated (Figure 5.9). Each page contains 10 search results. Users may navigate to the previous or the next page via the buttons at the bottom of the page. They may also slide the page to the top by selecting the arrow button on the bottom right.

The concept tree on the sidebar is interactive. A dialog would pop up when users select a concept (Figure 5.10). Users may set the filtering condition for



85 results found

3.1 INTRODUCTION

Book: Introduction to Digital Signal Processing and Filter Design

...*Fourier transform* ... of the discrete time system, and describe its properties and applications. In the second half of the chapter, we discuss another *transform* known as the discrete time *Fourier* series ... for periodic, discrete time signals. There is a third *transform* called discrete *Fourier transform* ... which is simply a part of the ... and we discuss its properties as well as its applications in signal processing. The use of ... to solve many of the problems or...

05 Windowing

Book: Digital Signal Processing: An Experimental Approach

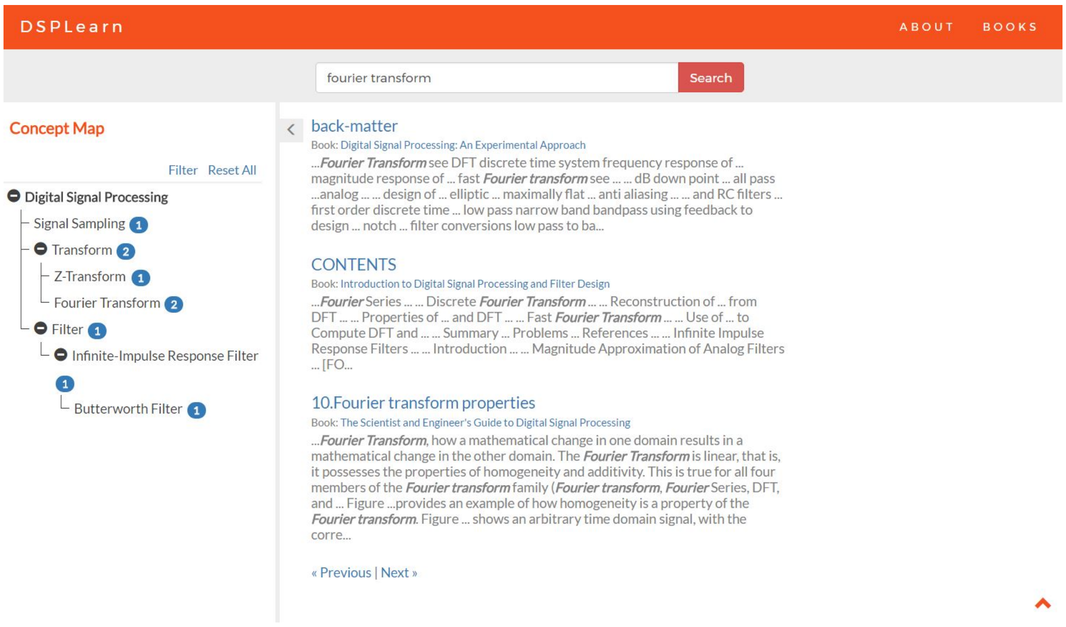
...*Fourier transform* of ... There is a second difficulty. Time limiting not only smears the *Fourier transform* around the correct frequency, but also causes the *Fourier transform* to extend across all frequencies. In addition to the central lobe, there are now additional lobes, ... as well. The smearing of the *Fourier transform* that is caused by time limiting a signal is commonly referred to as spectral leakage. We would like to cure the local smearing and the appearance of energy at freq...

04 Fourier Analysis and the Discrete Fourier Transform

Book: Digital Signal Processing: An Experimental Approach

...*Fourier transform*, we discuss its significance, and we derive its properties. Then we discuss the family of algorithms know as fast *Fourier transforms*. We explain their significance, describe how one uses them, and discuss zero padding and the fast convolution algorithm. Keywords. *Fourier transform*, discrete *Fourier transform*, fast *Fourier transform*, zero padding, fast convolution. Often, we need to determine the spectral content of a ... much of a signals power is located at a

Figure 5.8: DSPLearn Page for Search Results



back-matter

Book: Digital Signal Processing: An Experimental Approach

...*Fourier Transform* see DFT discrete time system frequency response of ... magnitude response of ... fast *Fourier transform* see ... ... dB down point ... all pass ...analog ... ... design of ... elliptic ... maximally flat ... anti aliasing ... ... and RC filters ... first order discrete time ... low pass narrow band bandpass using feedback to design ... notch ... filter conversions low pass to ba...

CONTENTS

Book: Introduction to Digital Signal Processing and Filter Design

...*Fourier* Series ... ... Discrete *Fourier Transform* ... ... Reconstruction of ... from DFT ... ... Properties of ... and DFT ... ... Fast *Fourier Transform* ... ... Use of ... to Compute DFT and ... ... Summary ... Problems ... References ... ... Infinite Impulse Response Filters ... ... Introduction ... ... Magnitude Approximation of Analog Filters ... [FO...

10.Fourier transform properties

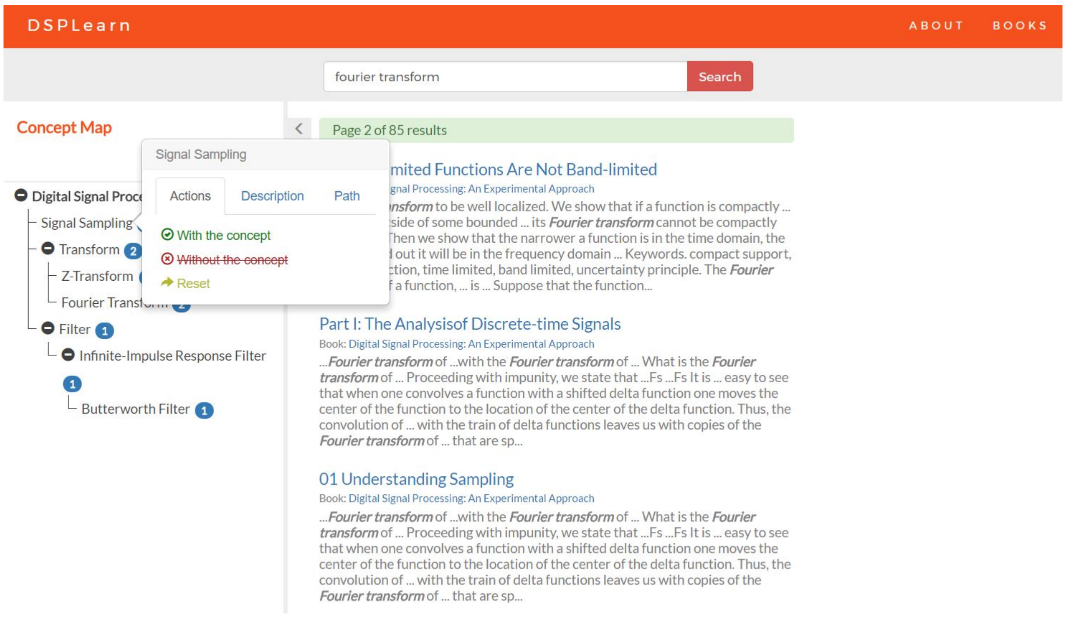
Book: The Scientist and Engineer's Guide to Digital Signal Processing

...*Fourier Transform*, how a mathematical change in one domain results in a mathematical change in the other domain. The *Fourier Transform* is linear, that is, it possesses the properties of homogeneity and additivity. This is true for all four members of the *Fourier transform* family (*Fourier transform*, *Fourier* Series, DFT, and ... Figure ...provides an example of how homogeneity is a property of the *Fourier transform*. Figure ... shows an arbitrary time domain signal, with the corre...

« Previous | Next »

Figure 5.9: Pagination of Search Results

each concept by selecting whether each document of the search results must or must not be related to the concept.



mited Functions Are Not Band-limited
gnal Processing: An Experimental Approach
*nsform* to be well localized. We show that if a function is compactly ... side of some bounded ... its ***Fourier transform*** cannot be compactly Then we show that the narrower a function is in the time domain, the out it will be in the frequency domain ... Keywords. compact support, ction, time limited, band limited, uncertainty principle. The ***Fourier*** f a function, ... is ... Suppose that the function...

Part I: The Analysisof Discrete-time Signals
Book: Digital Signal Processing: An Experimental Approach
...***Fourier transform*** of ...with the ***Fourier transform*** of ... What is the ***Fourier transform*** of ... Proceeding with impunity, we state that ...Fs ...Fs It is ... easy to see that when one convolves a function with a shifted delta function one moves the center of the function to the location of the center of the delta function. Thus, the convolution of ... with the train of delta functions leaves us with copies of the ***Fourier transform*** of ... that are sp...

01 Understanding Sampling
Book: Digital Signal Processing: An Experimental Approach
...***Fourier transform*** of ...with the ***Fourier transform*** of ... What is the ***Fourier transform*** of ... Proceeding with impunity, we state that ...Fs ...Fs It is ... easy to see that when one convolves a function with a shifted delta function one moves the center of the function to the location of the center of the delta function. Thus, the convolution of ... with the train of delta functions leaves us with copies of the ***Fourier transform*** of ... that are sp...

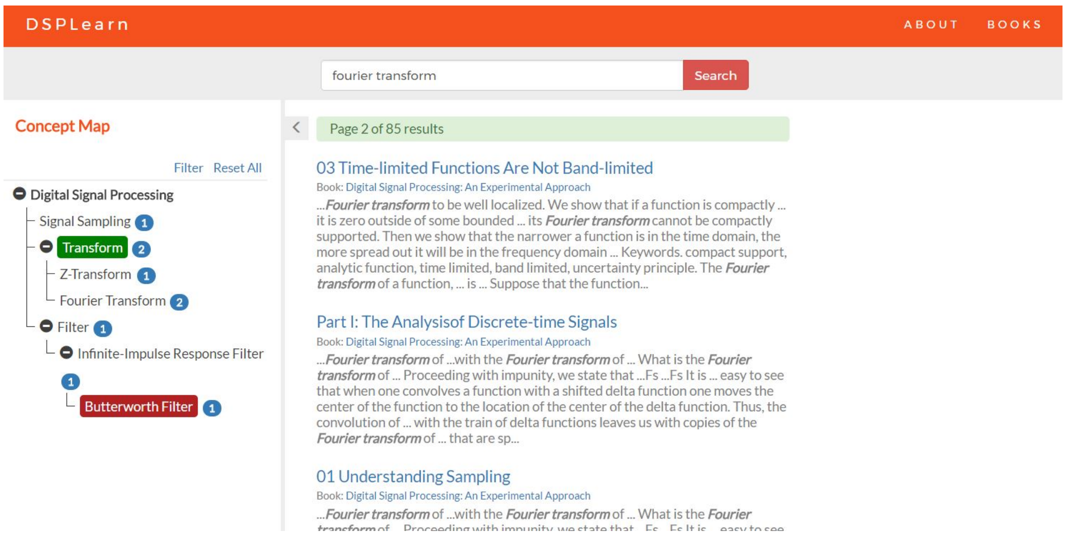
Figure 5.10: Concept Popover in Result Page

The dialog has 3 tabs - "Actions", "Description", and "Path". The description of each tab is below.

- **Actions:** Choose an action to filter the search results - "With the concept", "Without the concept", or "Reset"
- **Description:** Definition of the concept
- **Path:** Concept path from the root concept node to the current node in the ontology

As seen in Figure 5.11a, a concept would be in green if users select "With the concept" in the concept dialog and in red if users select "Without the concept" instead. Users may select "Reset" to impose no constraint on the search results for the concept.

When the "Filter" button is selected, DSPLearn would filter the search results based on the filtering conditions. Figure 5.11b shows the results after the filtering is performed. As seen, the concept tree on the sidebar is also updated to reflect the new search results.



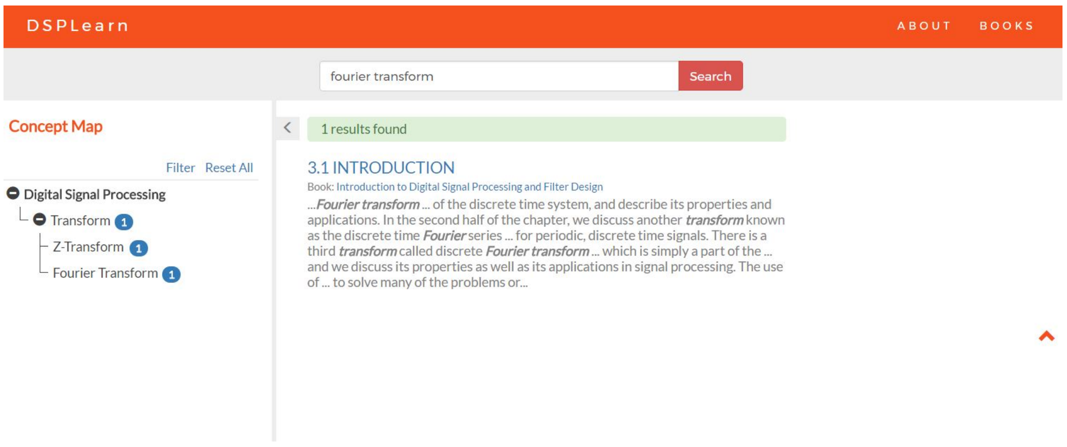
Page 2 of 85 results

03 Time-limited Functions Are Not Band-limited
Book: Digital Signal Processing: An Experimental Approach
...*Fourier transform* to be well localized. We show that if a function is compactly ... it is zero outside of some bounded ... its *Fourier transform* cannot be compactly supported. Then we show that the narrower a function is in the time domain, the more spread out it will be in the frequency domain ... Keywords. compact support, analytic function, time limited, band limited, uncertainty principle. The *Fourier transform* of a function, ... is ... Suppose that the function...

Part I: The Analysisof Discrete-time Signals
Book: Digital Signal Processing: An Experimental Approach
...*Fourier transform* of ...with the *Fourier transform* of ... What is the *Fourier transform* of ... Proceeding with impunity, we state that ...Fs ...Fs It is ... easy to see that when one convolves a function with a shifted delta function one moves the center of the function to the location of the center of the delta function. Thus, the convolution of ... with the train of delta functions leaves us with copies of the *Fourier transform* of ... that are sp...

01 Understanding Sampling
Book: Digital Signal Processing: An Experimental Approach
...*Fourier transform* of ...with the *Fourier transform* of ... What is the *Fourier*

(a) Before Filtering



1 results found

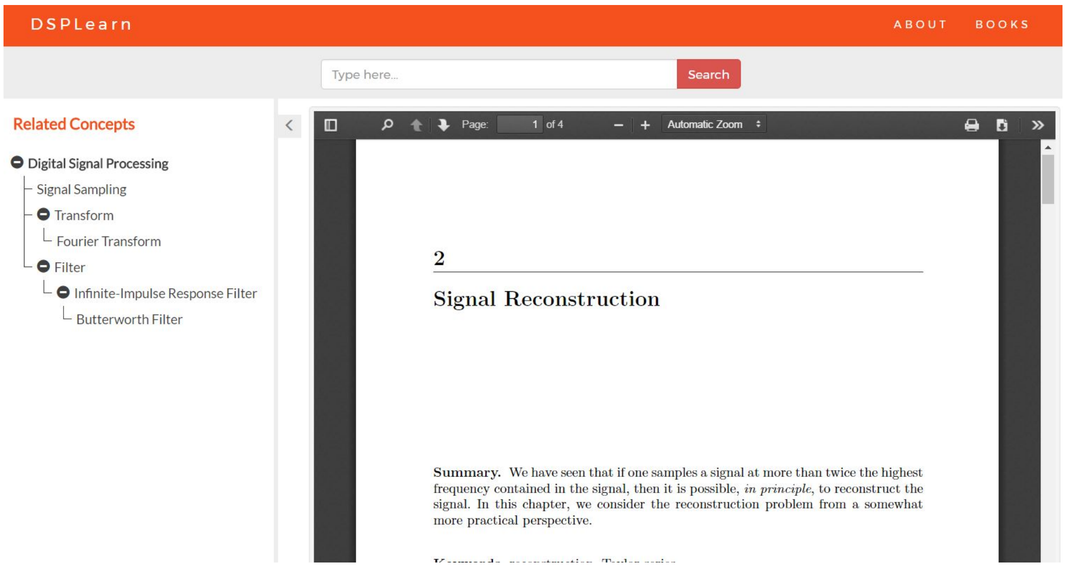
3.1 INTRODUCTION
Book: Introduction to Digital Signal Processing and Filter Design
...*Fourier transform* ... of the discrete time system, and describe its properties and applications. In the second half of the chapter, we discuss another *transform* known as the discrete time *Fourier* series ... for periodic, discrete time signals. There is a third *transform* called discrete *Fourier transform* ... which is simply a part of the ... and we discuss its properties as well as its applications in signal processing. The use of ... to solve many of the problems or...

(b) After Filtering

Figure 5.11: Search Results Filtering

## 5.5 Viewing

Figure 5.12 shows the detail page for DSPLearn. The detail page features a PDF viewer that supports an extensive range of PDF operations including print, download, zoom, rotate, search, and navigate. As seen, similar to the result page, the left sidebar also contains a hierarchical concept tree. The tree consists of concepts that are related to the section. There is also a search bar on top of the page.



2

Signal Reconstruction

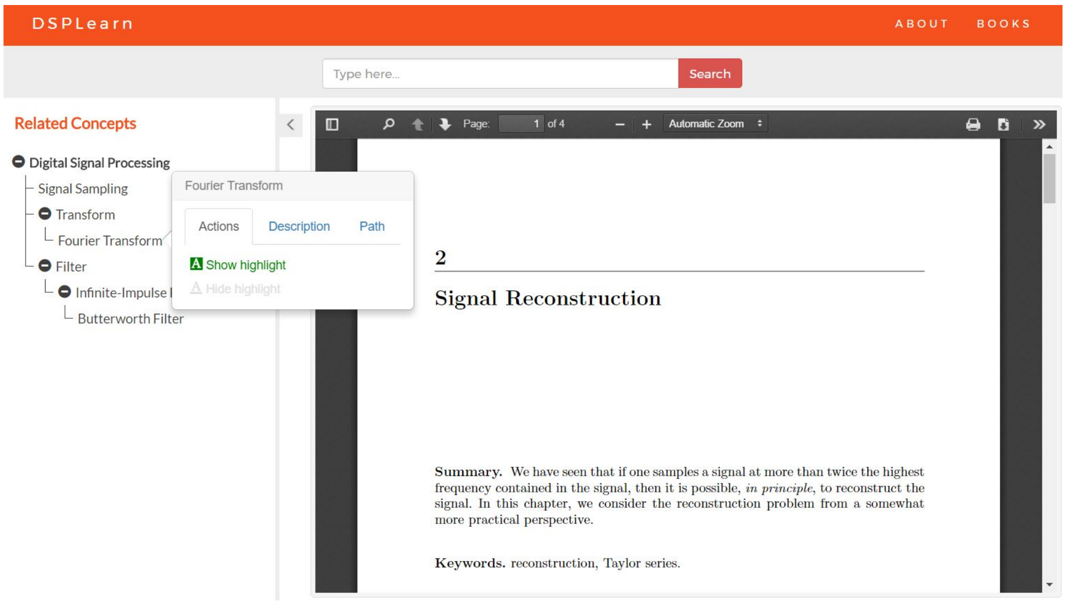
**Summary.** We have seen that if one samples a signal at more than twice the highest frequency contained in the signal, then it is possible, *in principle*, to reconstruct the signal. In this chapter, we consider the reconstruction problem from a somewhat more practical perspective.

Figure 5.12: DSPLearn Page for Viewing a Section

The concept tree on the sidebar is also interactive. A dialog would pop up when users select a concept (5.13). Users may choose to highlight terms in the PDF document that are mapped to the selected concepts in the tree.

The dialog also has 3 tabs - “Actions”, “Description”, and “Path”. There are 2 actions to choose from - “Show highlight” and “Hide highlight”. All terms are not highlighted by default. When “Show highlight” is chosen, the concept would be in green, as shown in Figure 5.14. When “Hide highlight” is chosen, the concept would revert back to normal.

In Figure 5.14, all terms mapped to the concept “Transform” are highlighted in the PDF document. The section ID for this document is 2192. Based on the ontology and term-to-concept mappings in Figure 5.6, only the second entry in the `Concept mappings` table is applicable. In the second entry, it says that the first, second, third, and fifth occurrences of the case-insensitive term “fourier transform” are mapped to the concept “Fourier Transform”. Since “Fourier Transform” is a child concept of “Transform”, as defined in the



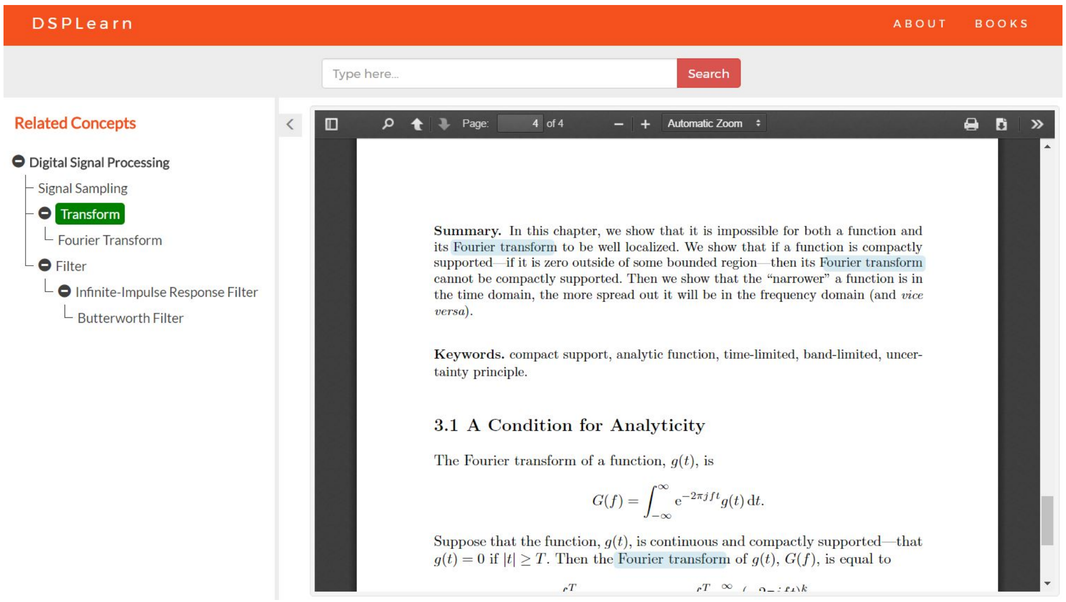
2

Signal Reconstruction

**Summary.** We have seen that if one samples a signal at more than twice the highest frequency contained in the signal, then it is possible, *in principle*, to reconstruct the signal. In this chapter, we consider the reconstruction problem from a somewhat more practical perspective.

**Keywords.** reconstruction, Taylor series.

Figure 5.13: Concept Popover in Detail Page



**Summary.** In this chapter, we show that it is impossible for both a function and its Fourier transform to be well localized. We show that if a function is compactly supported—if it is zero outside of some bounded region—then its Fourier transform cannot be compactly supported. Then we show that the "narrower" a function is in the time domain, the more spread out it will be in the frequency domain (and *vice versa*).

**Keywords.** compact support, analytic function, time-limited, band-limited, uncertainty principle.

3.1 A Condition for Analyticity

The Fourier transform of a function, $g(t)$, is

$$G(f) = \int_{-\infty}^{\infty} e^{-2\pi jft} g(t)\, dt.$$

Suppose that the function, $g(t)$, is continuous and compactly supported—that $g(t) = 0$ if $|t| \geq T$. Then the Fourier transform of $g(t)$, $G(f)$, is equal to

Figure 5.14: Term Highlighting in PDF

ontology (Figure 5.6a), these 4 occurrences of terms are also mapped to the concept “Transform”.

The PDF snippet in Figure 5.14 shows the highlights of the second, third, and fifth occurrences of the term “fourier transform”. As noted, the fourth occurrence of the term is not highlighted as expected.

# Chapter 6

# Conclusions and Future Work

## 6.1 Conclusions

An information retrieval web application in the domain of digital signal processing was developed in this project. This IR application, named DSPLearn, is a proof-of-concept prototype for keyword search with concept ontology.

DSPLearn can serve as a framework for information retrieval with ontology in a specific domain. The technologies used in DSPLearn are generic and can in general be applied to any kind of text and any kind of knowledge bases.

DSPLearn supports efficient search of PDF documents. It can generate a concept tree based on the search results for a query, from which users can filter the documents returned. It allows highlighting of terms that are mapped to user-selected concepts in a PDF document. The $n$-th match approach was proposed for locating an exact term in a PDF document.

DSPLearn was built on some work done by other undergraduate students - including a RESTful back-end for retrieval of document data and a program for removing formulas in text.

DSPLearn was designed and implemented with re-usability in mind. It uses readily available up-to-date technology solutions for the development of mobile-friendly and responsive modern web application. The code is highly modular and achieves low coupling and high cohesion. DSPLearn is well documented so that future developers can easily take over and make it a ready-to-use product.

## 6.2 Recommendations for Future Work

Due to the large scale of the system, some areas are not fully examined and there are some limitations in the current system. Future work can explore the following directions:

1. Ontology development for a domain of discourse;
2. Design and implementation of a concept recognition algorithm;
3. Functionality improvement of the IR web application.

### 6.2.1 Ontology Development

A simplified DSP ontology was manually crafted and stored in MySQL database for this project. Future researchers can work on the development of ontology for a specific domain, including the development process, the ontology life cycle, the methodologies for building ontology, and the tool suites and languages that support them.

The vision of the Semantic Web is to extend principles of the Web from documents to data. Ontology is the backbone to add rich semantics to the corresponding resources. It would gain increasing importance as the Web is involving into 3.0.

### 6.2.2 Concept Recognition

A traditional and simple concept recognition algorithm maps every occurrence of a term to a concept. However, this algorithm is undesirable because even same terms but in different locations may map to different concepts in practice. A context-aware concept recognition algorithm based on machine learning is thus needed so that a term in a specific location can be semantically mapped to concepts.

In this project, the design and implementation of concept recognition algorithm is not the concerns. Term-to-concept mappings are manually added into the MySQL database for demonstration purpose.

### 6.2.3 Functionality Improvement

The IR application for this project was designed to provide an information retrieval framework with ontology. It follows the open/closed software principle and allows future developers to add on more features.

Developers may enrich the IR web application with more user-friendly functionalities, such as:

- Suggest similar documents to users based on the documents users are viewing;
- Add a comments module to allow users to post comments when viewing documents;
- Support more interactive visualization to show the linkage among documents.

# Bibliography

[1] What is big data? `https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html`. [Online; accessed 5-Mar-2017].

[2] David Shenk. *Data smog : surviving the information glut.* HarperCollins Publishers, 1997. ISBN 00601870189780060187019.

[3] Derek Dean and Caroline Webb. Recovering from information overload. `http://www.mckinsey.com/business-functions/organization/our-insights/recovering-from-information-overload`, January 2011. [Online; accessed 5-Mar-2017].

[4] Bhaskar Kapoor and Savita Sharma. A comparative study ontology building tools for semantic web applications. *International journal of Web & Semantic Technology*, 1(3), July 2010.

[5] Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web. *Scientific American Magazine*, 2001.

[6] Thomas R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[7] Ontology editors. `https://www.w3.org/wiki/Ontology_editors`, February 2015. [Online; accessed 6-Mar-2017].

[8] Web ontology language (owl). `https://www.w3.org/2001/sw/wiki/OWL`, December 2012. [Online; accessed 6-Mar-2017].

[9] Ralph Delfs, Andreas Doms, Alexander Kozlenkov, and Michael Schroeder. Gopubmed: ontology-based literature search applied to gene ontology and pubmed. *Nucleic Acids Research*, July 2005. doi: 10.1093/nar/gki470.

[10] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing and Management*, 24(5): 513–523, 1988.

[11] Fatiha Boubekeur and Wassila Azzoug. Concept-based indexing in text information retrieval. *International Journal of Computer Science & Information Technology*, 5(1), February 2013.

[12] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN 9780521865715.

[13] Why django? `https://www.djangoproject.com/start/overview/`, . [Online; accessed 13-Mar-2017].

[14] Writing your first django app, part 1. `https://docs.djangoproject.com/en/1.10/intro/tutorial01/`, . [Online; accessed 15-Mar-2017].

[15] Celery - distributed task queue. `http://docs.celeryproject.org/en/3.1/`. [Online; accessed 15-Mar-2017].

[16] Introduction to redis. `https://redis.io/topics/introduction`. [Online; accessed 15-Mar-2017].

[17] Solr. `http://lucene.apache.org/solr/`. [Online; accessed 15-Mar-2017].

[18] Models. `https://docs.djangoproject.com/en/1.10/topics/db/models/`, . [Online; accessed 13-Mar-2017].

[19] Welcome to haystack! `http://django-haystack.readthedocs.io/en/v2.5.1/`. [Online; accessed 15-Mar-2017].

[20] django-treebeard. `http://django-treebeard.readthedocs.io/en/latest/index.html`. [Online; accessed 15-Mar-2017].

[21] Pdf.js. `https://mozilla.github.io/pdf.js/`, . [Online; accessed 15-Mar-2017].

[22] Pdf.js. `https://en.wikipedia.org/wiki/PDF.js`, . [Online; accessed 15-Mar-2017].

[23] jquery. `https://jquery.com/`. [Online; accessed 15-Mar-2017].

# Appendices

# Appendix A

# RESTful API of Document Server

| Group | Parent URL | Endpoint | Child URL | Method | Description |
|---|---|---|---|---|---|
| Book | /book | List | /list | GET | List all the e-books. Each book contains book ID, book title, and ID of the root section. |
| | | Detail | /detail/{pk}/ | GET | Return details of a book specified by {pk} (i.e. book ID), including book ID, book title, and ID of the root section. |

Continued on next page

Continued from previous page

| | | | | | |
|---|---|---|---|---|---|
| | | TOC | /toc/{pk}/ | GET | Return the table of contents of a book specified by {pk}. It is a nested and recursive JSON that represents the table of contents tree. |
| | | PDF | /read/{pk}/[{start}/{end}/] | GET | Display the PDF of the book specified by {pk}, starting from page specified by {start} and ending with page specified by {end}. If no starting and ending pages are specified, it will return the whole book in PDF. |
| Section | /section | Detail | /detail/{pk}/ | GET | Return details of a section specified by {pk} (i.e. section ID), including section ID, section title, and a Boolean value specifying whether the section has child sections. |
| | | Children | /children/{pk}/ | GET | Return section details for all child sections of a section specified by {pk}. |

Continued on next page

Continued from previous page

| | | | | | |
|---|---|---|---|---|---|
| | | Versions | /versions/{pk}/ | GET | Return an array of all the versions associated with the section specified by {pk}. |
| | | Partial TOC | /toc/{pk}/ | GET | Return the partial TOC starting from the section specified by {pk}. |
| Version | /version | List | /list | GET | List all the versions. Each version contains version ID, version name, version creator, and timestamp. |
| | | Detail | /detail/{pk}/ | GET | Return details of a version specified by {pk} (i.e. version ID), including version ID, version name, version creator, and timestamp. |
| | | Create | /create/ | PUT | Create a new version. Only version name needs to be specified by user. |
| | | Update | /update/{pk}/ | POST | Update version name of the version specified by {pk}. |
| | | Delete | /delete/{pk}/ | DELETE | Delete a version specified by {pk}. |

Continued on next page

Continued from previous page

| | | | | | |
|---|---|---|---|---|---|
| Content | /content | Immediate Text | /immediate/{section}/[{version}/] | GET | Return the text of a certain version specified by {version} (i.e. version ID) of a section specified by {section} (i.e. section ID). If no version is specified, the raw version will be chosen by default. |
| | | Aggregate Text | /aggregate/{section}/{version}/ | GET | Return the text of a certain version specified by {version} (i.e. version ID) of a section specified by {section} (i.e. section ID) and its all descendants. If no version is specified, the raw version will be chosen by default. |
| | | Post | /post/{section}/{version}/ | POST | The body of the request contains the location of the text file with content of a specific version, specified by {version}, of a section, specified by {section}. |

# Appendix B

# Environment Setup for Application Development

The environment setup procedures for the development of the IR web application are presented here. The procedures described below are based on Windows 10 host platform, with Intel Core i7-4700HQ processor, 8 GB RAM, and 1 TB HDD. For other operating systems or other system specifications, users may follow similar procedures to set up the environment but specific settings or installation may vary.

## B.1 Install Ubuntu Virtual Machine

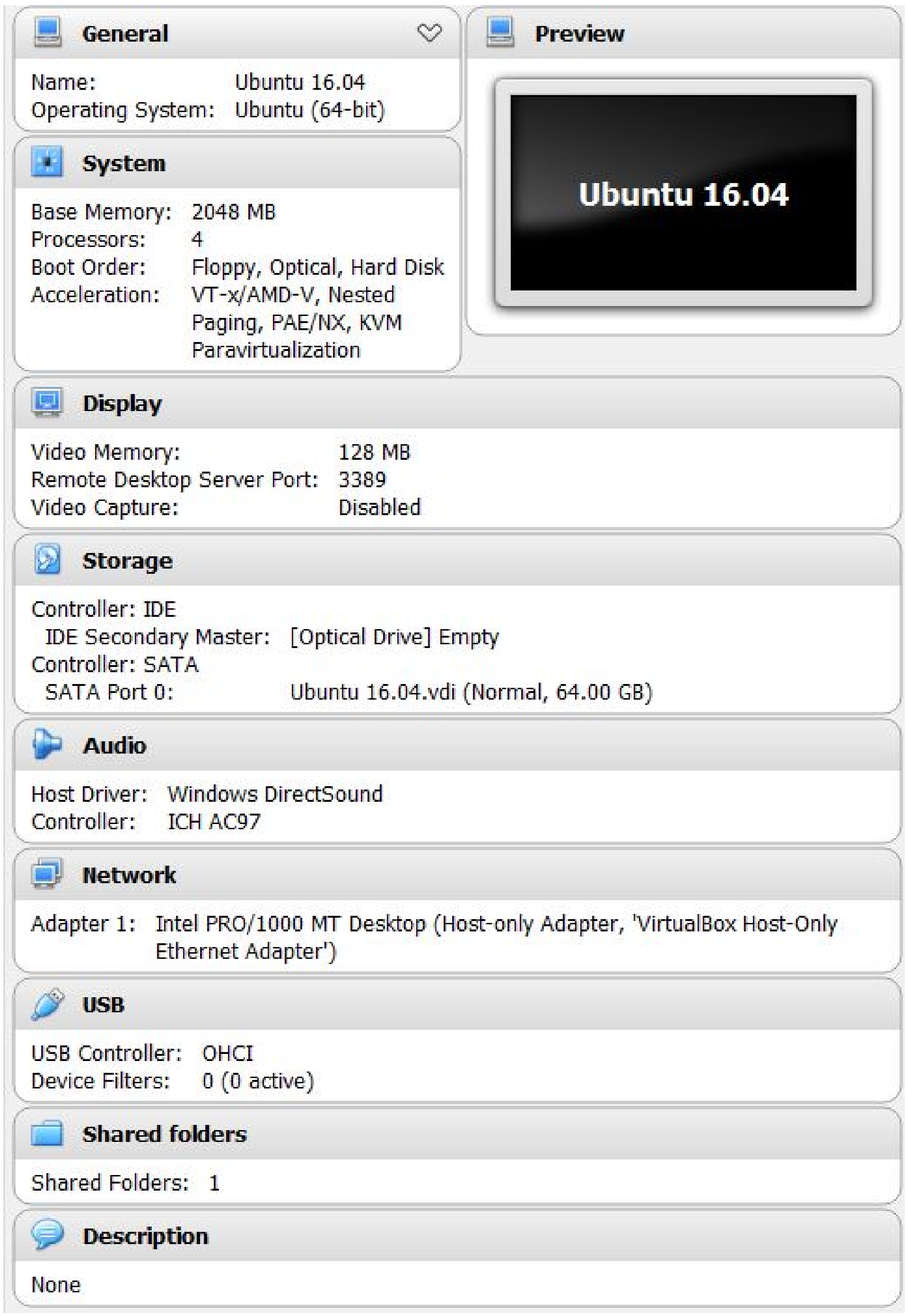
### B.1.1 Install VirtualBox

Visit VirtualBox's download page (`https://www.virtualbox.org/wiki/Downloads`). Select the Windows installer and install the latest version in the same way as any normal Windows programs.

### B.1.2 Install Ubuntu

Get a Ubuntu disk image (.iso file) from Ubuntu's download page (`https://www.ubuntu.com/download/desktop`). Next, install Ubuntu inside Windows using VirtualBox. You may refer to instructions in `http://www.psychocats.net/ubuntu/virtualbox`.

Ubuntu 16.04 was used for the development. For your reference, Figure B.1 is the author's settings for the virtual machine. The settings should be applicable to most of the commercial laptops.



General

| | |
|---|---|
| Name: | Ubuntu 16.04 |
| Operating System: | Ubuntu (64-bit) |

Preview

System

| | |
|---|---|
| Base Memory: | 2048 MB |
| Processors: | 4 |
| Boot Order: | Floppy, Optical, Hard Disk |
| Acceleration: | VT-x/AMD-V, Nested Paging, PAE/NX, KVM Paravirtualization |

Display

| | |
|---|---|
| Video Memory: | 128 MB |
| Remote Desktop Server Port: | 3389 |
| Video Capture: | Disabled |

Storage

| | |
|---|---|
| Controller: IDE | |
| IDE Secondary Master: | [Optical Drive] Empty |
| Controller: SATA | |
| SATA Port 0: | Ubuntu 16.04.vdi (Normal, 64.00 GB) |

Audio

| | |
|---|---|
| Host Driver: | Windows DirectSound |
| Controller: | ICH AC97 |

Network

| | |
|---|---|
| Adapter 1: | Intel PRO/1000 MT Desktop (Host-only Adapter, 'VirtualBox Host-Only Ethernet Adapter') |

USB

| | |
|---|---|
| USB Controller: | OHCI |
| Device Filters: | 0 (0 active) |

Shared folders

| | |
|---|---|
| Shared Folders: | 1 |

Description

None

Figure B.1: Ubuntu Virtual Machine Configurations

During development, we need to access the guest machine from the host. Thus, host-only networking is used to create a private Ethernet network consisting of the host and the guest only. You may identify the IP address of the virtual machine using the command:

```
hostname -I
```

The IP address is likely to be 192.168.56.101. The Django application server would need to access the Ubuntu virtual machine via this IP address. However, there is no Internet access for host-only networking; thus, you may need to change to NAT networking when you perform Step B.2.

## B.2 Host Solr, Redis, and MySQL in Ubuntu

### B.2.1 Host Solr

Boot up the Ubuntu virtual machine. Solr comes in a pre-packaged form that requires very little other than JRE and Jetty. Install Solr by running the following commands from a terminal prompt:

```
curl -LO https://archive.apache.org/dist/lucene/solr/4.10.2/solr-4.10.2.tgz
tar xvzf solr-4.10.2.tgz
cd solr-4.10.2/example/
mv solr/collection1/ solr/dsp_collection/
```

Make ensure that you are in the directory `./solr-4.10.2/example`. Start Solr server by executing the following command:

```
java -jar start.jar
```

You may now access Solr admin page from the host machine using the URL `http://<guest_ip_address>:8983/solr`.

### B.2.2 Host Redis

Install Redis by running the following commands from a terminal prompt:

```
wget http://download.redis.io/redis-stable.tar.gz
tar xvzf redis-stable.tar.gz
cd redis-stable
make
```

Make ensure that you are in the directory `./redis-stable`. Start Redis server by executing the following command:

```
redis-server redis.conf
```

### B.2.3 Host MySQL

Install MySQL by running the following commands from a terminal prompt:

```
sudo apt-get update
sudo apt-get install mysql-server
```

During the installation process you will be prompted to enter a password for the MySQL root user. Once the installation is complete, the MySQL server should be started automatically. You can run the following command from a terminal prompt to check whether the MySQL server is running:

```
service mysql status
```

If the server is not running correctly, you can type the following command to start it:

```
sudo service mysql restart
```

## B.3 Create Python Virtual Environment in Windows

### B.3.1 Install Python 3

You may install the latest Python 3 version from Python's download page (`https://www.python.org/downloads/`). The Python package manager `pip` is included in Python 3.4 and above by default.

To access `pip` from any directory in the Command Prompt, make sure the `Scripts` subdirectory of Python is in the environment variable `PATH`. For example, if Python is installed in `C:\ProgramFiles(x86)\Python\Python35-32\`, you should make sure `C:\ProgramFiles(x86)\Python\Python35-32\Scripts` is in `PATH`.

### B.3.2 Install Virtual Environment Packages

`virtualenv` is a tool to create isolated Python environments, each with their own libraries and site-packages. `virtualenvwrapper` is a set of commands which makes working with virtual environments much more pleasant. Both packages should be installed into the same global site-packages area inside Python's directory. You may need administrative privileges to do that. Install the two packages by running the following commands from Windows Command Prompt:

```
pip install virtualenv
pip install virtualenvwrapper-win
```

`virtualenvwrapper` places all virtual environments in one place. By default, they are placed in `%USERPROFILE%\Envs`.

### B.3.3 Create Python Virtual Environment

Create a new virtual environment named `<name>` by running the following command in the Command Prompt:

```
mkvirtualenv <name>
```

Activate the environment named `<name>` by running the following command:

```
workon <name>
```

You may deactivate the working environment and switch back to the global environment by running the command:

```
deactivate
```

## B.4 Clone Project from GitHub

### B.4.1 Install Python IDE PyCharm

PyCharm was the sole integrated development environment (IDE) for the entire project, including front-end and back-end development. It is strongly recommended because of its extensive developer-friendly features.

You may install PyCharm Professional from PyCharm's download page (`https://www.jetbrains.com/pycharm/download/`). If you are student, you may get free JetBrains license with your school email. More information is here: `https://www.jetbrains.com/student/`.

### B.4.2 Install Git

Visit Git's download page (`https://git-scm.com/downloads`) and install the latest version of Git. Specify the location of the Git executable file on the `Git` page of the `Settings` dialog box in PyCharm.

### B.4.3 Clone Project from GitHub

Log in to your GitHub account in the `GitHub` page of the `Settings` dialog box in PyCharm.

Next, we need to check out the project from GitHub. Choose `Checkout from Version Control | GitHub` in PyCharm. Set the Git Repository URL to be `https://github.com/HarveyLeo/myfyp.git`. You may set your local project location in the same dialog. Finally choose `Clone`. The source code is now cloned into your local directory.

## B.5 Install Required Python Packages

Before you are ready to go, you need to install all the required python packages specified in `requirements.txt`. These are the packages necessary for the installation of technology stack mentioned in Section 3.5.

Whenever you work on the project, make sure you are in the Python virtual environment created in Step B.3.3. You can go to the virtual environment by running the `workon` command, as mentioned earlier. After going to the environment, navigate to the project home directory `myfyp/` by using `cd` command in PyCharm Terminal. Next, install all required packages by running the following command:

```
pip install -r requirements.txt
```

You may list all unused dependencies by running the command below and remove them using `pip uninstall`.

```
pip-autoremove --list
```

## B.6 Configure Solr and MySQL

### B.6.1 Update Ubuntu's IP Address in Settings

Check Ubuntu's IP address and update the address in `settings.py`, which is located in the project directory `myfyp/myfyp/`. The IP address is set to

192.168.56.101 by default. You may replace all occurrences of this IP address in `settings.py` with Ubuntu virtual machine's actual IP address.

### B.6.2 Configure Solr Schema

Generate Solr's `schema.xml` from Haystack by running the following command in PyCharm Terminal:

```
python manage.py build_solr_schema > schema.xml
```

Make sure you are in the Python virtual environment named `<name>` before issuing the command above. Place the generated `schema.xml` inside the directory `solr-4.10.2/example/solr/dsp_collection/conf/`. Then restart Solr.

### B.6.3 Configure MySQL Database

#### Create Remote User Account and Database

To grant remote access of MySQL database from any IP address, we need to create a user account in MySQL. Log in to the root account from a terminal prompt in the virtual machine using the command:

```
mysql -u root -p
```

Enter the password that you created earlier for the MySQL root user. Once successfully logging in, run the following commands:

```
CREATE USER 'mysql-client'@'%' IDENTIFIED BY 'password';
GRANT ALL PRIVILEGES ON *.* TO 'mysql-client'@'%' WITH GRANT OPTION;
FLUSH PRIVILEGES;
```

A new MySQL user account named `mysql-client` is hence created. The asterisks in the second command refers to databases and tables respectively that the account can access. This specific command allows the account to perform all tasks across all databases and tables. You may narrow the access privileges for this account later. To tell the server to reload the grant tables, a flush-privileges operation is performed by issuing the last command above.

Next, create a database named `dsp_database` by running the command below in the `mysql>` prompt:

```
CREATE DATABASE dsp_database;
```

### Change to UTF-8 Encoding

To display the current character encoding set for the database `dsp_database`, run the following command at the `mysql>` prompt:

```
SELECT default_character_set_name FROM information_schema.SCHEMATA
WHERE schema_name = "dsp_database";
```

If the character encoding is not UTF-8, convert to UTF-8 by running the following command at the `mysql>` prompt:

```
ALTER DATABASE dsp_database CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

This does not convert existing tables. It only sets the default for newly created tables. However, at this stage there is no table in the database `dsp_database`.

### Migrate Changes from Django Models

You can now update the MySQL database named `dsp_database` by running the following commands in PyCharm Terminal:

```
python manage.py makemigrations dsp_index
python manage.py migrate
```

By running `makemigrations`, you are telling Django that you have made some changes to your models and that you would like the changes to be stored as a *migration*. The `migrate` command takes all the migrations that have not been applied and applies those changes to the database.

## B.7 Start All Servers

Before running the web application, you need to start Redis, Solr, and MySQL in Ubuntu; then, start Django application server and Celery worker server in PyCharm Terminal in Windows. When you are in the Python virtual environment named `<name>`, navigate to the project home directory `myfyp/` and start the application server by running the following command:

```
python manage.py runserver
```

Open a new Terminal in PyCharm, go to the virtual environment, and start the Celery worker server by running the following `celery` command with `worker` argument:

```
celery -A myfyp worker -l info
```

`myfyp` is the current Django project name and `-l` is to indicate the log level.

## B.8 Access Webpages through Browsers

You are now ready to go! The homepage, django-admin page, dsplearn-admin page can be accessed respectively via the following URLs.

- Homepage: `http://127.0.0.1:8000/`
- Django-admin page: `http://127.0.0.1:8000/django-admin/`
- Dsplearn-admin page: `http://127.0.0.1:8000/admin/`

To log in to the django-admin page, you need to create a superuser - a user account that has control over everything on the site. Type the following command to create a new superuser in PyCharm Terminal:

```
python manage.py createsuperuser
```

You would need to crawl document data from the Document Server and index into Solr before searching. Go to the dsplearn-admin page and crawl data now! It took approximately 1 hour to crawl 1000 sections. Once crawling is finished, you may build the search index in Solr following commands in Section 4.3.2.

# Appendix C

# Django Models

```
from django.db import models
from treebeard.mp_tree import MP_Node
from django.core.validators import validate_comma_separated_integer_list


class Book(models.Model):
    book_id = models.IntegerField(primary_key=True)
    title = models.CharField(max_length=255)
    pages = models.IntegerField()
    pdf = models.CharField(max_length=100)

    def __str__(self):
        return "book {id}: {title}"\
            .format(id=self.book_id, title=self.title)


class Section(models.Model):
    section_id = models.IntegerField(primary_key=True)
    title = models.CharField(max_length=255)
    text = models.TextField()
    book = models.ForeignKey(Book, on_delete=models.CASCADE)
    page = models.IntegerField()
    pdf = models.CharField(max_length=100)

    def __str__(self):
        return "section {id}: {title}"\
            .format(id=self.section_id, title=self.title)


class Concept(MP_Node):
    label = models.CharField(max_length=20, unique=True)
    name = models.CharField(max_length=255)

    node_order_by = ['label', 'name']

    def __str__(self):
        return "{0} {1}".format(self.label, self.name)


class ConceptMapping(models.Model):
    concept = models.ForeignKey(Concept, on_delete=models.CASCADE)
    term = models.CharField(max_length=255)
    section = models.ForeignKey(Section, on_delete=models.CASCADE)
    nth_match = models.CharField(max_length=100,
                validators=[validate_comma_separated_integer_list])

    def __str__(self):
        return "concept mapping: ({sid}, {nth}, {term})->{concept}"\
            .format(sid=self.section.section_id,
                    nth=self.nth_match,
                    term=self.term,
                    concept=self.concept.name)

    class Meta:
        unique_together = ('concept', 'section', 'term',)
```

# Appendix D

# Ontology Creation via Python Shell

Since there is no existing DSP ontology yet, you may manually create a simplified ontology in the MySQL database via Python shell.

Set up the development environment as described in Appendix B. Change the working virtual environment to the one you created during environment setup. Enter Python shell by running the following command in PyCharm Terminal:

```
python manage.py shell
```

Create an ontology with 13 hierarchical concepts in the MySQL database with the following script:

```python
from dsp_index.models import Concept
get = lambda node_id: Concept.objects.get(pk=node_id)
root = Concept.add_root(label='0', name='Digital Signal Processing')
get(root.pk).add_child(label='1', name='Signal Sampling')
node2 = get(root.pk).add_child(label='2', name='Transform')
node3 = get(root.pk).add_child(label='3', name='Filter')
get(node2.pk).add_child(label='2.1', name='Z-Transform')
get(node2.pk).add_child(label='2.2', name='Fourier Transform')
get(node2.pk).add_child(label='2.3', name='Laplace Transform')
node31 = get(node3.pk).add_child(label='3.1',
                                 name='Finite-Impulse Response Filter')
node32 = get(node3.pk).add_child(label='3.2',
                                 name='Infinite-Impulse Response Filter')
get(node31.pk).add_child(label='3.1.1', name='Moving Average Filter')
get(node32.pk).add_child(label='3.2.1', name='Butterworth Filter')
get(node32.pk).add_child(label='3.2.2', name='Elliptic Filter')
get(node32.pk).add_child(label='3.2.3', name='Chebyshev Filter')
Concept.dump_bulk()
```

You may see the ontology just created by checking the table `Concepts` in the django-admin page. Now, you may create some term-to-concept mappings by adding entries to the table `Concept mappings` in the django-admin page.