CS410 Project Report

Search Engine for Video Segment Search of CS410 Lecture Videos

1 Team Information

Team name: Torrey Pines
Team member (NetID): Wusi Chen (wusic2)
Team captain: Wusi Chen
Project work contribution: Wusi Chen (100% contribution)

2 Introduction and Work Overview

In this project, a search engine application is built that can support the segment search of CS410 course video on Coursera. With this search engine, learners of this course can find the relevant segments of course lecture videos to their queries, and can play the lecture video from the beginning of the segments in Coursera. This can improve the learning experience of this course, since this search engine can save learner's time in watching unrelated course videos before they find the relevant video segments.

Although Coursera has the search bar on the top of their webpages which can find the relevant video segments, that search feature will return the only the video segments which match all the query terms. For example, the search feature in Coursera will return no video segment if the query is "bag of word entropy", as shown in Figure 1. This means the search engine for video segment search can only return the video segments whose transcript matches all the query terms. This unannounced restriction/limitation will deteriorate the learning experience, since as the people who do not have a full understanding of the course material, the learners may not know if the terms in their queries do not happen at the same time in a segment of the transcript.

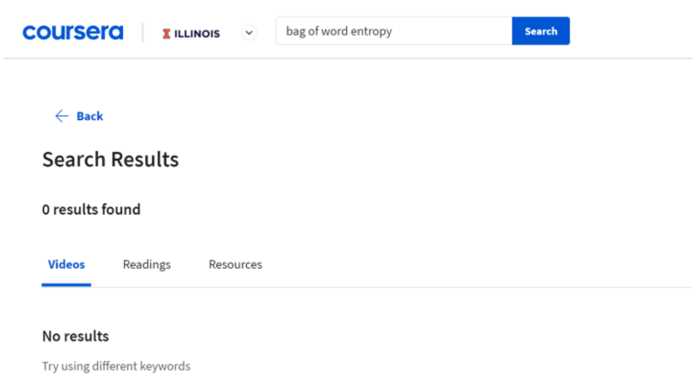


Figure 1 Search result of "bag of word entropy" from Coursera

The work of this project can be divided into 2 parts. The first part is to build a Python program which processes the subtitle files (vtt files) downloaded from Coursera, segments the lecture videos, and stores the data of video segments in a format which is compatible for Metapy library to rank relevant video segments, and for the search engine application to retrieve the information of the relevant segments. The second part is to build a Python-based web application to provide the learners a user-friendly interface to enter queries related to this course, and to see a list of relevant video segments, and to provide links to play the lecture video from the beginning of the video segments in Coursera.

3 Application Usage
3.1 Installation

To install the search engine application for video segment search, Python 3.7 shall be installed to the machine, and the project files can be cloned from GitHub using the following command:

```
git clone https://github.com/chenwusi2012/torrey_pines
cd .\torrey_pines
```

The following Python libraries shall be installed to run the project code:

- webvtt-py: To parse the subtitle files for lecture videos downloaded from Coursera.
- Metapy: To create inverted index for the segment transcripts, and to rank the relevant segment transcripts based on the query.
- Pandas: To manipulate the data of all the video segments, and to save or load the data from CSV files.
- pytoml: To read the file config.toml in the project directory.
- Bottle: To provide the framework for search engine website.

The txt file requirements.txt is provided in the project directory. To install the above Python libraries automatically, run the following command under the project directory:

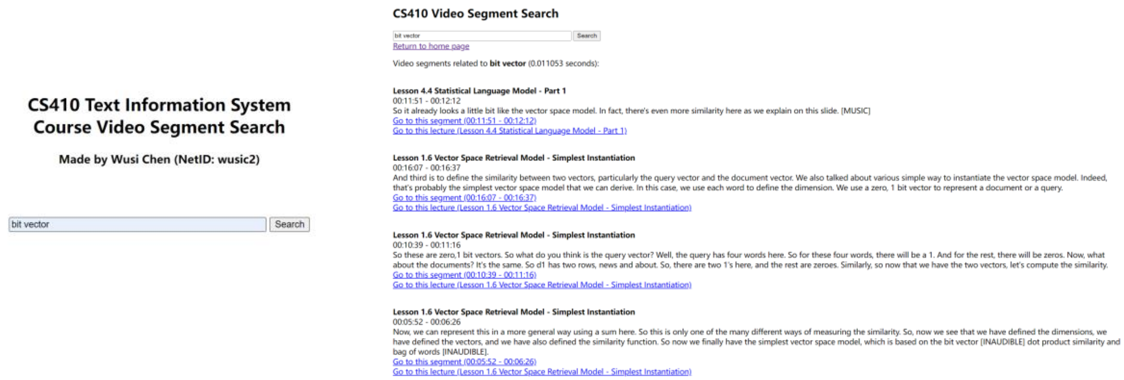
```
pip install -r requirements.txt
```

To launch the search engine application (website), run the following command under the project directory:

```
python ./bottle_app.py.
```

3.2 Use

After the launch of the search engine application, the users can go to the home page of the search engine application/website through this URL: http://localhost:8080/. If the search engine application/website is deployed to a public URL, the users can enter the home page using the public URL. After entering the home page, the users can use the application like other search engines, such as Google. In the home page, the users can type the keywords for search in the input box, as shown in Figure 2 (a). After clicking the search button, the users can see a list of relevant video segments with the lecture name, start timestamp, end timestamp, and transcript, as shown in Figure 2 (b). At the end of each video segment, 2 hyperlinks are provided. By clicking the link “Go to this segment,” the users can go to the corresponding video segment in Coursera, and the time of the lecture video in Coursera is set to the start timestamp of the segment. By clicking the link “Go to this lecture”, the users can go to the lecture video in Coursera, and play the lecture video from the beginning. From the page of search result, the users can start another video segment search by entering a new query to the input box at the top of the webpage, and clicking the search button.



(a) **(b)**

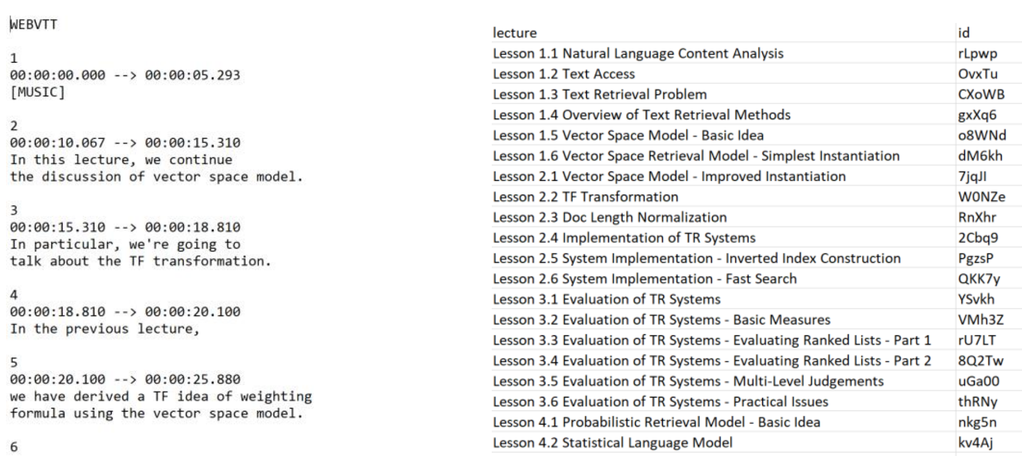
Figure 2 (a) Home page of search engine website of CS410 video segments (b) Sample search result.

4 Implementation

4.1 Parser (parser.py)

The purpose of the parser is to process the subtitle files (WebVTT files) for CS410 lecture videos, and to save the data into the file format which is compatible to Metapy library and the search engine application. The transcript in the subtitle files is separated into segments. The length of the segment is defined by the time duration of the corresponding video segment. Each segment is extracted and saved when the time duration is no less than a threshold (The current setting is 30s).

There are 2 types of input files for the parser. The first type of input files is the subtitles files (vtt files) downloaded from Coursera website and saved in the subtitle folder. The screenshot of a sample subtitle file is shown in Figure 3 (a). The second type of the input files is link_info.csv in the dataset folder. This CSV file is manually prepared, and has 2 columns. One is the name of all the lecture videos, which is also the names of all the subtitle files (vtt files), and the other is the identification code of all the lecture videos as shown in Figure 3 (b). The link of a lecture video is "https://www.coursera.org/learn/cs-410/lecture/" followed by the identification code of that video.



```
WEBVTT

1
00:00:00.000 --> 00:00:05.293
[MUSIC]

2
00:00:10.067 --> 00:00:15.310
In this lecture, we continue
the discussion of vector space model.

3
00:00:15.310 --> 00:00:18.810
In particular, we're going to
talk about the TF transformation.

4
00:00:18.810 --> 00:00:20.100
In the previous lecture,

5
00:00:20.100 --> 00:00:25.880
we have derived a TF idea of weighting
formula using the vector space model.

6
```

lecture	id
Lesson 1.1 Natural Language Content Analysis	rLpwp
Lesson 1.2 Text Access	OvxTu
Lesson 1.3 Text Retrieval Problem	CXoWB
Lesson 1.4 Overview of Text Retrieval Methods	gxXq6
Lesson 1.5 Vector Space Model - Basic Idea	o8WNd
Lesson 1.6 Vector Space Retrieval Model - Simplest Instantiation	dM6kh
Lesson 2.1 Vector Space Model - Improved Instantiation	7jqJl
Lesson 2.2 TF Transformation	W0NZe
Lesson 2.3 Doc Length Normalization	RnXhr
Lesson 2.4 Implementation of TR Systems	2Cbq9
Lesson 2.5 System Implementation - Inverted Index Construction	PgzsP
Lesson 2.6 System Implementation - Fast Search	QKK7y
Lesson 3.1 Evaluation of TR Systems	YSvkh
Lesson 3.2 Evaluation of TR Systems - Basic Measures	VMh3Z
Lesson 3.3 Evaluation of TR Systems - Evaluating Ranked Lists - Part 1	rU7LT
Lesson 3.4 Evaluation of TR Systems - Evaluating Ranked Lists - Part 2	8Q2Tw
Lesson 3.5 Evaluation of TR Systems - Multi-Level Judgements	uGa00
Lesson 3.6 Evaluation of TR Systems - Practical Issues	thRNy
Lesson 4.1 Probabilistic Retrieval Model - Basic Idea	nkg5n
Lesson 4.2 Statistical Language Model	kv4Aj

(a) **(b)**

Figure 3 (a) Sample subtitle file (Lesson 2.2 TF Transformation); (b) Head of link_info.csv

There are 2 output files from the parser. The first output file is data.csv in the dataset folder. This CSV file contain the information of all the parsed video segments, including their start and end timestamps, lecture name, lecture identification code, and transcript as shown in Figure 4. The second output file is cs410.dat file in the cs410 folder. Each line in this file corresponds to the transcript of one video segment. This file is for creating the inverted index through Metapy library.

	lecture	lecture_id	start_time	end_time	text
0	Lesson 1.1 Natural Langu	rLpwp	0:00:00	0:00:32	[SOUND] >> This lecture is about Natural Language of Content Analysis. As you see from this picture, this is reall
1	Lesson 1.1 Natural Langu	rLpwp	0:00:32	0:01:05	We're going to cover three things. First, what is natural language processing, which is the main technique for pr
2	Lesson 1.1 Natural Langu	rLpwp	0:01:06	0:01:42	Now what do you have to do in order to understand that text? This is basically what computers are facing. So lo
3	Lesson 1.1 Natural Langu	rLpwp	0:01:42	0:02:15	So that's the first step. After that, we're going to figure out the structure of the sentence. So for example, here i
4	Lesson 1.1 Natural Langu	rLpwp	0:02:16	0:02:51	So here we show we have noun phrases as intermediate components, and then verbal phrases. Finally we have a
5	Lesson 1.1 Natural Langu	rLpwp	0:02:51	0:03:23	For example, you might imagine a dog that looks like that. There's a boy and there's some activity here. But for a
6	Lesson 1.1 Natural Langu	rLpwp	0:03:25	0:03:56	Now from this representation we could also further infer some other things, and we might indeed naturally thin
7	Lesson 1.1 Natural Langu	rLpwp	0:03:56	0:04:29	You can even go further to understand why the person say at this sentence. So this has to do as a use of languag
8	Lesson 1.1 Natural Langu	rLpwp	0:04:29	0:05:01	That could be one possible intent. To reach this level of understanding would require all of these steps and a cor
9	Lesson 1.1 Natural Langu	rLpwp	0:05:01	0:05:33	Computers unfortunately are hard to obtain such understanding. They don't have such a knowledge base. They a
10	Lesson 1.1 Natural Langu	rLpwp	0:05:33	0:06:05	For example, programming languages. Those are harder for us, right? So natural languages is designed to make
11	Lesson 1.1 Natural Langu	rLpwp	0:06:05	0:06:36	We could overload the same word with different meanings without the problem. Because of these reasons this
12	Lesson 1.1 Natural Langu	rLpwp	0:06:39	0:07:13	The word of root may have multiple meanings. So square root in math sense or the root of a plant. You might be
13	Lesson 1.1 Natural Langu	rLpwp	0:07:16	0:07:54	So this is an example of synaptic ambiguity. What we have different is structures that can be applied to the sam
14	Lesson 1.1 Natural Langu	rLpwp	0:07:55	0:08:27	Another example of difficulty is anaphora resolution. So think about the sentence John persuaded Bill to buy a T
15	Lesson 1.1 Natural Langu	rLpwp	0:08:27	0:09:00	It would have to use a lot of knowledge to figure that out. It also would have to maintain a large knowledge bas
16	Lesson 1.1 Natural Langu	rLpwp	0:09:01	0:09:33	We can do part of speech tagging pretty well, so I showed 97% accuracy here. Now this number is obviously bas

Figure 4 Head of data.csv

The parser will do the following actions (The flowchart is shown in Figure 10 (a) in Appendix **on the last page**):

- Create a Pandas dataframe for collecting parsed segments.
- Load link_info.csv as a Pandas dataframe.
- Iterate the rows in this Pandas dataframe to iterate the subtitles of all the lecture videos.
- For each row, read the lecture name and lecture identification code, and find the corresponding vtt file in the subtitle folder.
- Read the found vtt file using webvtt-py library. This library reads the captions in the vtt file line by line.
    - For each line, get the start timestamp, end timestamp, and caption. If the caption does not stop at a period, read the next line. Save the start timestamp as the start timestamp of a segment.
    - Keep reading the next lines. Accumulate the transcript for this segment. Keep updating the end timestamp of this segment.
    - If one line stops at a period, check the time difference between the start timestamp and end timestamp of the segment. If the difference is less than the threshold (the current setting is 30 seconds), keep reading more lines until the next period. If the difference is more than the threshold, save the lecture name, lecture identification code, start timestamp, end timestamp, and transcript to the created Pandas dataframe. Besides, save the transcript of the segment to a line in cs410.dat.
- After iterating all the rows in the dataframe from link_info.csv, save the dataframe with all the segment information to data.csv.
- Create an inverted index for all the segments in cs410.dat using Metapy (remove the idx folder if the inverted index already exists).

After running the parser (parser.py), 96 lectures are processed (96 vtt subtitle files), and the data of 2188 video segments are stored in 2 output files.

4.2 Search Engine Application (bottle_app.py)

The purpose of the search engine application is to provide a user-friendly interface for searching course video segments related to users' queries. The website for search engine application is built based on the Python micro web-framework Bottle (Initially, Flask was chosen as the web-framework for search engine application. However, the application will get stuck and run forever when it calls "ranker.score" method). In bottle_app.py, the functions with a route() decorator are callback functions, as shown in Figure 5. When a URL path inside route() is called, the corresponding callback function is executed.

```
@route('/')
def root():
    print("Home page")
    return template('home')


@route('/search', method='POST')
def search():
    a = datetime.datetime.now()
    is_empty = False
    print("Result page")
    query = request.forms.get('query')
    query = query.strip()
```

Figure 5 Callback Function in bottle_app.py

When a user calls the URL path "/", and the callback function "root" is called, which renders the home page with the template home.tpl in the views folder. This template has a form HTML element which can collect the user's query through the input box. When the user clicks the search button, the URL path "/search" and the corresponding callback function "search" are called, which leverages Metapy library to find relevant video segments to the user's query, and renders the template result.tpl to display the information of the relevant segments on the webpage.

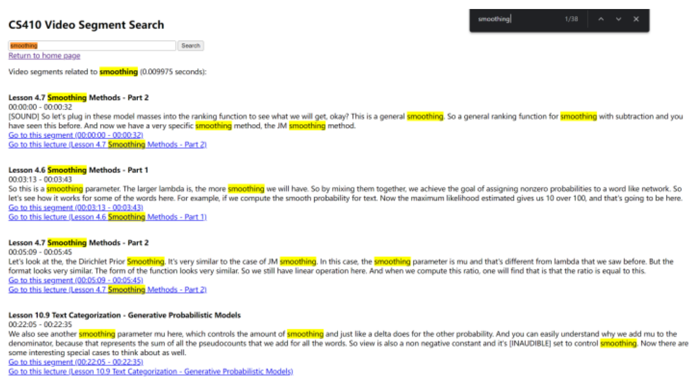
In detail, the callback function "search" will do the following actions (The flowchart is shown in Figure 10 (b) in Appendix **on the last page**):

- Load data.csv in the dataset folder to a Pandas dataframe (this is done when the application is launched).
- Get the query from the form from the home.tpl.
- Call the defined "rank" function. This function leverages Metapy library to return a list of the indexes for relevant video segments.
- Using the returned indexes, collect the full information of the relevant segments. For each of the relevant segments:
  - Get the lecture name, lecture identification code, start timestamp, end timestamp, transcript of the segment.
  - Form the link of the lecture video, which is "https://www.coursera.org/learn/cs-410/lecture/" followed by the identification code of that video.
  - Form the link to the video segment, which is "https://www.coursera.org/learn/cs-410/lecture/" followed by the identification code of that video, a string "?t=", and the start time of the segment in second. The start time is calculated through the function "convert_time".
  - Save the information of the segment as a dictionary "segment".

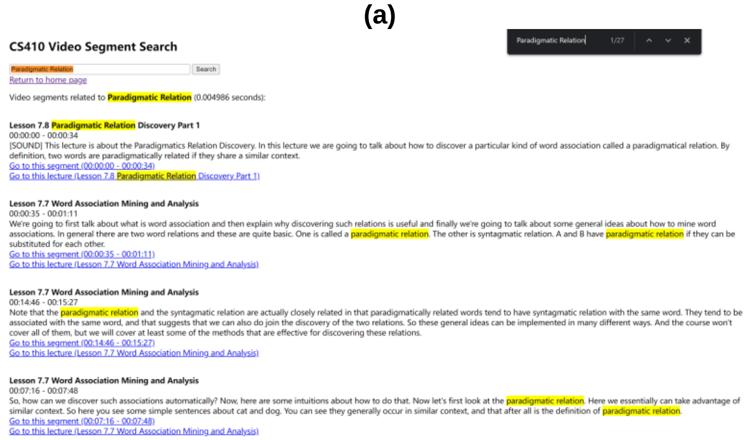
- Append the dictionary “segment to the list “result”, which contains the information of all the relevant segments.
- Render the result webpage with the template result.tpl. Pass the string “query” the list “result” to the template.

5 Demonstration of Functionality

The screenshots of video segment search result for several queries are shown in Figure 6. The query terms are highlighted by the web browser (Ctrl + F, and type the query). From the screenshots, we can see that the search engine application can return the relevant video segments whose transcript contains the query terms.



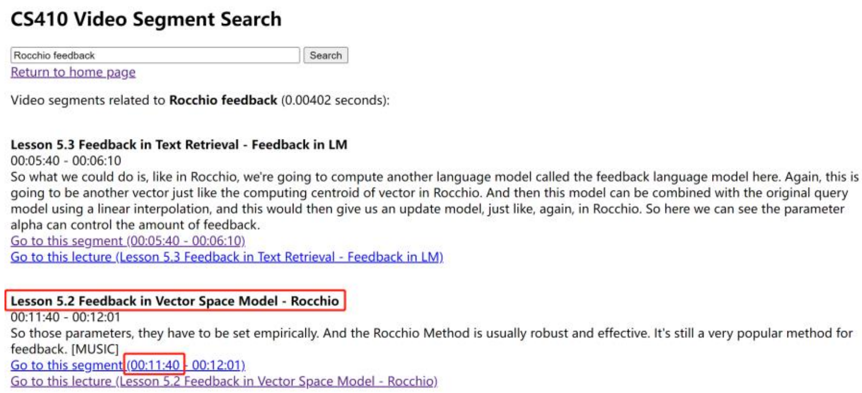
(a)



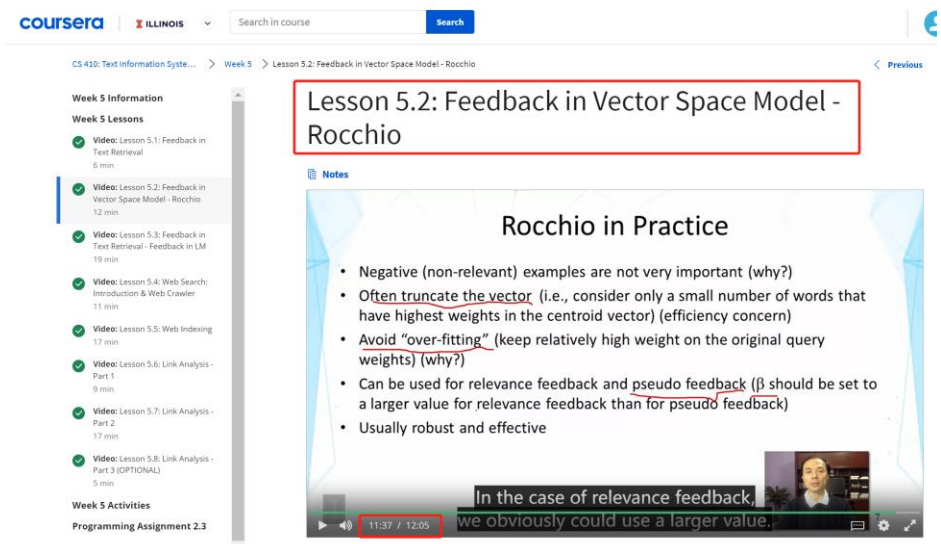
(b)

Figure 6 (a) Search result for query “smoothing”; (b) Search result for query “paradigmatic relation”. Query terms highlighted by web browser

As shown in Figure 7, when the users click the link “Go to this segment” for a video segment, the corresponding lecture video is displayed, and the start timestamp of the video player is set to the start timestamp of the corresponding video segment minus 3 seconds (Coursera minus 3 seconds from the start timestamp to make sure the viewer can see the start of the video segment without missing any transcript).



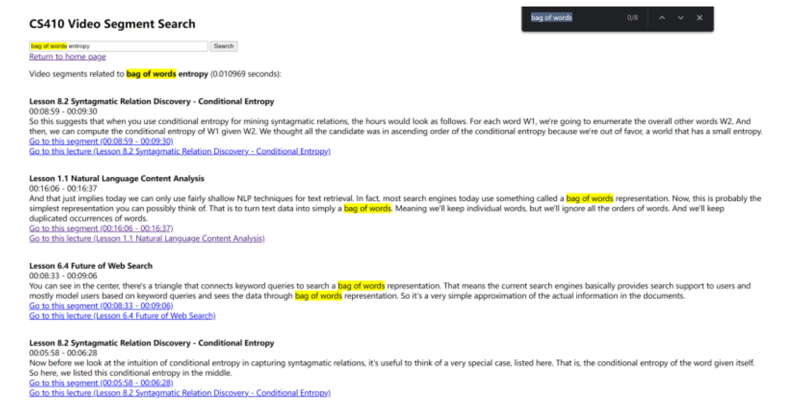
(a)



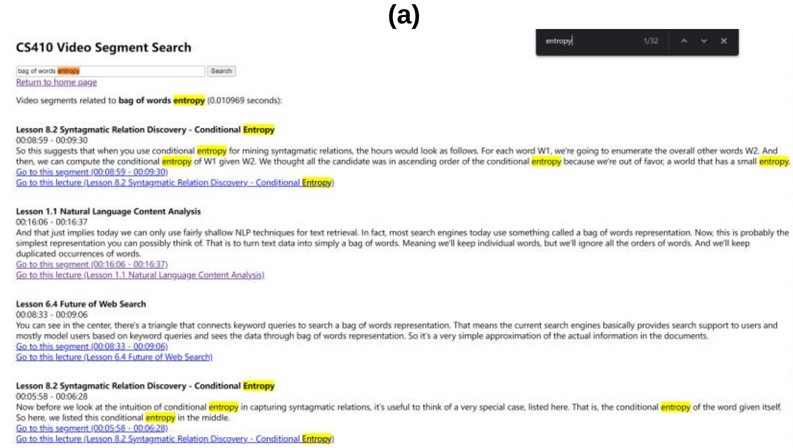
(b)

Figure 7 (a) Search result for query "Rocchio"; (b) Lecture video in Coursera after clicking link to second video segment

As mentioned in Section 2, the search engine in Coursera can only return the video segments whose transcript contains all the query term. For example, and the engine will return no result when the query term is "bag of words entropy". The same query is tested on the search engine application developed in this project. As shown in Figure 8, the developed engine can return the relevant video segments relevant to "bag of word" or "entropy". This shows that the developed engine is capable to return the result for non-related query terms.



(a)



(b)

Figure 8 (a) Search result for query "bag of words entropy" with highlight of "bag of words"; (b) Search result for query "bag of words entropy" with highlight of "entropy"

6 Performance Evaluation

To further evaluate the performance of the search engine application, a small set of queries is selected and used as the test cases to test the application. For each of the selected queries, the relevance of the top 10 returned segments is manually judged, and the precision@10 is calculated. The recall of the selected test cases is not evaluated, since the evaluation of recall needs the manual relevance judgement of all the segments in the collection (2188 segments) against the queries. The queries, precision@10, and average precision are listed in Table 1.

Table 1 Precision of Set of Queries

Query	Precision@10
semantic analysis	0.9
syntactic analysis	0.9
ambiguity	0.9
pos tagging	0.7
bag of words	1
pull mode	0.8
push mode	0.8
search engine	1
recommender system	1
bm25	1

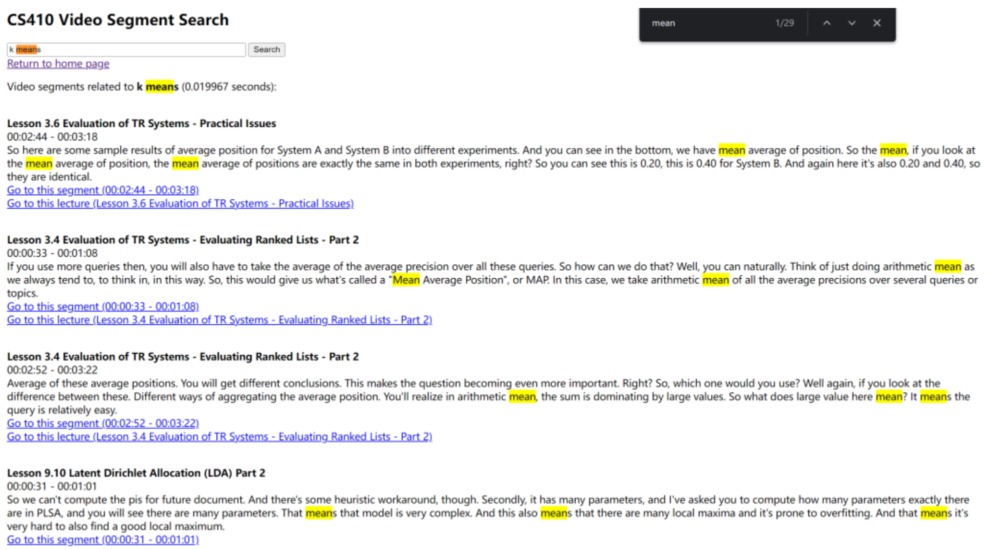
Query	Precision@10
ranking function	1
Vector space model	1
IDF Weighting	1
TF transformation	0.7
Inverted index	1
Precision	0.9
Recall	1
Average precision	0.8
Query Likelihood	1
Unigram Language Model	1
Smoothing	1
Rocchio Feedback	1
KL Divergence	0.8
Map reduce	0.7
Page rank	0.8
Content based filtering	0.5
Threshold Learning	1
Collaborative Filtering	1
Paradigmatic relation	1
Syntagmatic relation	1
Conditional entropy	1
Mutual information	1
Maximum likelihood	1
Posterior probability	1
Topic model	0.7
Background model	1
Mixture model	1
EM algorithm	1
PLSA	1
LDA	1
Text clustering	1
K Means	0
Naïve bayes	1
Logistic regression	1
Opinion mining	1
Sentiment classification	0.8
Causal Topic Modeling	1
Average Mean	0.9085

As shown in Table 1, the developed search engine application can effectively return the relevant video segments to the queries in the top 10 returned segments. However, the precision is 0 when the query is “K means”, which means the none of the top 10 video segments returned by the application are related to the concept of “K means”. The partial result is shown in Figure 9 (a). As shown in Figure 9 (a), this issue is because the word “mean” has the following word-level ambiguity:

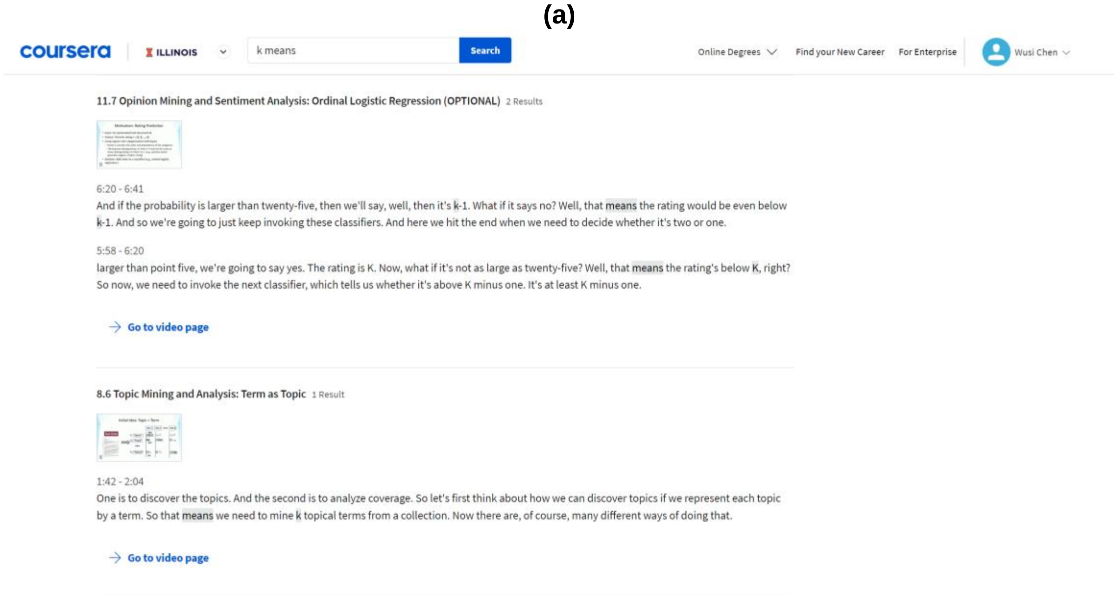
- The word “mean” can be a noun (average in math).

- The word “mean” can be a verb (explaining or clarifying a term or a sentence).

The query “k mean” is also run on the search engine in Coursera. The partial result is shown in Figure 9 (b). The issue is suggested be resolved by using 2-gram of word when running Metapy library.



(a)



(b)

Figure 9 (a) Result of “k mean” from developed search engine. (b) Result of “k mean” from search engine in Coursera.

Appendix

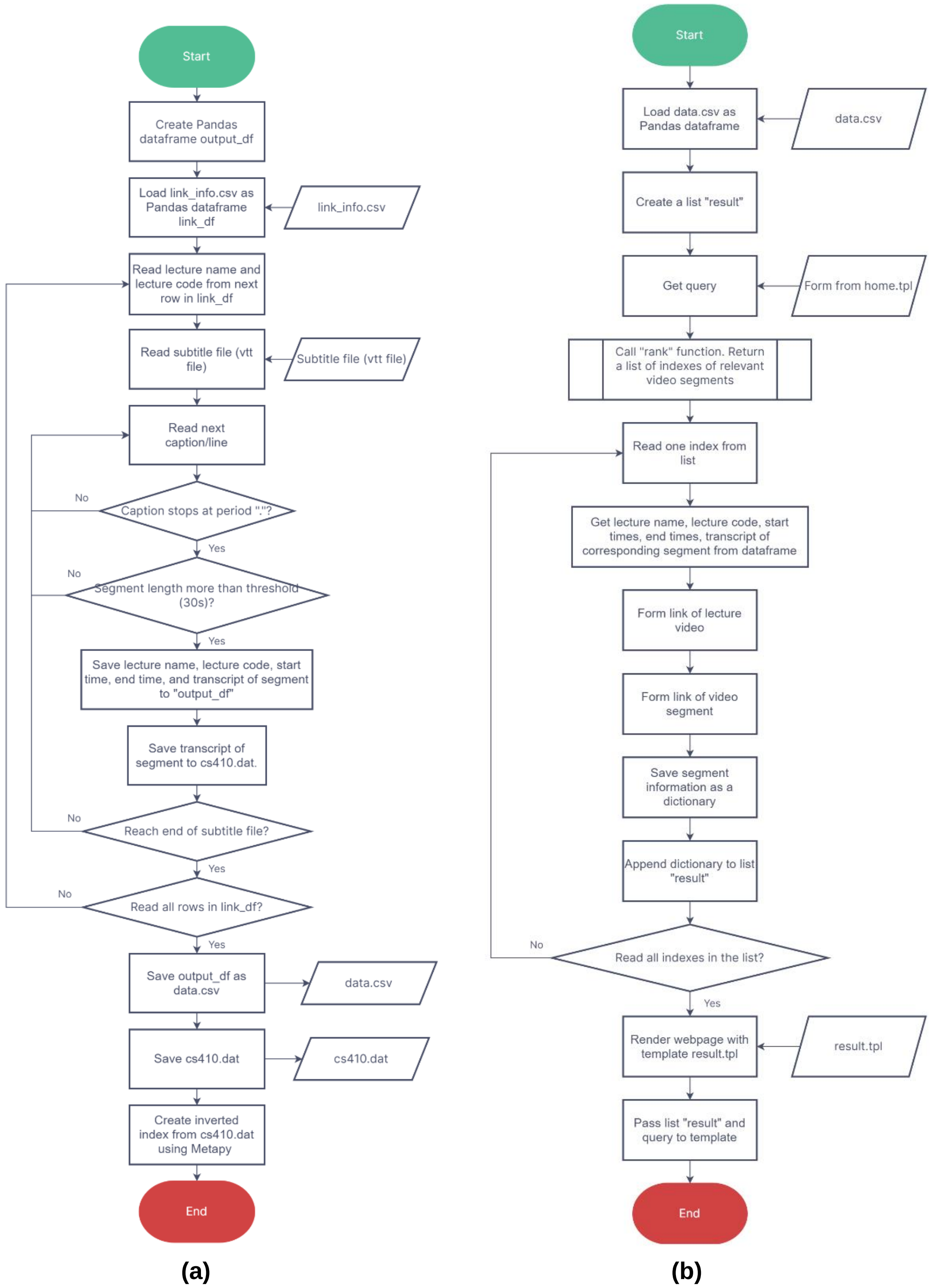


Figure 10 (a) Flowchart for parser (parser.py); (b) Flowchart for "search" callback function (in bottle_app.py)