1. INTRODUCTION

Spoken tutorials are a free resource of audio-video lectures and demonstrations about various open source software. It’s an initiative by IIT Bombay to popularize free and open source software development. Additionally it provides an efficacious platform for beginners to grasp the basics of open source software. The website consists of multiple videos by topic. These videos range between five and twelve minutes approximately in duration and are ordered based on difficulty and level of the topics. These videos explain in brief the foundations of the concept from the platform used, to the installation of necessary software.

Code demonstrations along with simple assignments related to the topic taught provide a practical exposure to the individual. A platform for discussion and resolution of issues is provided in the form of a forum within the website. The full text of audio is available in the form of timed scripts along with slides of the topic and source code of all programs which were executed in the tutorial. This enables a user to try out the sample codes for themselves and use it as a guideline to work on the exercises given.

Since these videos provide only a brief demonstration of any topic a user will inadvertently have doubts or will require information to further enhance domain knowledge. Further, a user will require more details regarding syntax and formats of the coding language as well as some reference to tackle problem definition. For a beginner, which these tutorials are aimed at, the availability of additional information is an essential feature.

There is a vast amount of information available on the internet and users exploring a new subject are increasingly becoming keen on learning more. As the information content on the internet is exponentially increasing, the user is surrounded by enormous amount of content. The task of pinpointing a specific location for consistent information and reducing multiple sources of the same text from various locations is an arduous and time consuming task which can hinder the process of actually gaining new information of value.

Online forums are an understated source of data. These forums contain a vast amount of information about a topic in the form of answers to a question, replies to an answer and comments. Further, most forums have myriads of questions, making the task of searching for a relevant post for information tedious in addition to the obvious possibility of repetitive posting. Within the noise in forum threads, a lot of the answers contain applicable and useful information.

Information should be quick and easy to access, and searching for information is a time consuming task. Hence, a technique to gather relevant information and present it concisely to a user is a feature of utmost importance.

We develop a mechanism to help such users. While watching a video, suppose the user wants to garner more information about the video he's watching. We want to minimize the amount of effort invested by the user in doing so.

The core of the project is similarity. An effective technique which can compare the video transcript to the forum posts is aimed to be developed if the user is to be provided with the most accurate results which can be of any guidance.

Therefore, we assist the user watching those videos by providing him/her with relevant forum links. These forum links can be from various forums including Stackoverflow, Stackexchange, etc.

While the user/student is watching the video, links to these forums that talk about the content being taught in the video are displayed alongside the video. The user can navigate to these forums when he doesn't fully understand the video lectures.

These links are displayed at regular intervals while the video is playing. Every few seconds, a new set of links that are relevant to the topic being discussed in the video are displayed. This way, a wide range of links are provided to the user/student over the course of the video. These links are displayed in the decreasing order of their relevance to the video.

A user friendly interface has been developed for the convenience of the user. User can choose the topic and the corresponding Spoken Tutorial video along with the links are displayed.

Therefore, in this way, the amount of time and effort spent by the user in searching for topics across the internet is immensely reduced while making content availability easier to him/her.

1.1 PROBLEM DEFINITION

It is well known that seeing and hearing someone explain a process greatly improves understanding. This is the reason for the enormous amount of MOOCs these days available online. Spoken Tutorials is one such course which helps users to learn and use open source software.

Even though these courses are quite thorough, the practicals are often very different than the theory part of any course. Thus, while taking a course in any such domain, a user is bound to encounter roadblocks either in understanding the content or while implementing the same.

Our aim is to collate relevant information from various threads or forums with similar content matter and present the most germane solution with respect to a particular time interval in the video to the viewer. The solutions presented are in the form of forum links ranked in their order of relevance.

The advantage of this is twofold. One being that these forum links try to solve the problems a user might face while getting his hands dirty before he even thinks about it. The other is that the user is able to explore beyond the limited areas the course is able to cover due to the constraint of time, bandwidth, etc.

1.2 PROPOSED SOLUTION

The problem requires a large dataset for the model to be trained on. Data is collected manually from prominent forum websites and tagged according to their content matter as relevant, slightly relevant and irrelevant. Since this project is to be demonstrated on a select few tutorial topics before further extension, the choice of topics has been chosen where sufficient data from forum links can be obtained with relative ease. The topics narrowed down are Advance C, C and Cpp and Bash.

The timed scripts are scraped from the website and these transcripts need to be matched against data to extract a measure of similarity. The concept of similarity is the centre of this project. The technique which can provide the best possible similarity value, taking into consideration the size of dataset and the difference in language of the transcript snippet which is formal and forum language which is informal should be developed. A study on various techniques and procedures on existing methods is recorded. This is to benchmark the said solutions against each other to gain a view of the technique most suitable for this project. Using this information a model is created by building upon a selected algorithm and the accuracy is noted.

Each mapping, to obtain the similarity is done between a time interval of a video from the timed scripts and the matching forum link from which data (forum text) has been obtained.

Every thread is composed of posts in the form of comments and replies to answers. These threads are decomposed into their components in the form of a bag of posts and matched with each other. A weight will then be assigned to each post based on their number of upvotes to rank them in accordance to their relevance.

An extensive timeline is provided later in this document but the estimated time of delivery is targeted at the last week of April 2017.

2. LITERATURE SURVEY

- Commonly used techniques for text comparison include mathematical formulae like cosine similarity and Euclidean distances[1]
  - The paper discusses the existing techniques on text similarity through partitioning them into three approaches, String-based, Corpus-based and Knowledge-based similarities. Furthermore, samples of combination between these similarities are presented.
- Clustering techniques can be efficiently used to seek, manage and integrate the huge volume of contents being generated in different forums[2]
  - It discusses three clustering methods: hierarchical agglomerative clustering, k-Means, and probabilistic latent semantic analysis
  - However, this method is not a feasible design choice for the model due to the time taken to process data into clusters.
- Finding similar threads by decomposing threads into set of weighted overlapping components[3]
  - This paper addresses the problem of finding similar threads to a given thread by proposing a novel methodology of decomposing the thread into a set of weighted overlapping components. It then estimates pairwise thread similarities by quantifying how well the information in the threads are mutually contained within each other using lexical similarities between their underlying components.
  - Basically, a thread is treated as a bag of posts, wherein the scoring function ise posed as an aggregate of the pairwise similarity between posts.
  - The paper describes four substructures, *the entire thread, single posts, post-reply pairs,* and *the dialogue.*
  - However, we only consider the *single posts* structure which is the only structure which closely resembles our model.

- Reasoning: The paper is trying to find the similarity between different threads, so they need to take into consideration the posts along with the reply to those posts
- However, we are interested only in the similarity between the transcript and the entire thread. So we just want the posts which are direct reply to the question of the thread.

[1] Gomaa, Fahmy (2013). A Survey of Text Similarity Approaches.

[2] Pattabiraman, Sodhi, Zhai (2013). Exploiting Forum Thread Structures to Improve Thread Clustering.

[3] Amit, Deepak, Dinesh (2012). Retrieving Similar Discussion Forum Threads.

3. SYSTEM REQUIREMENTS SPECIFICATION

3.1 HIGH LEVEL BLOCK DIAGRAM

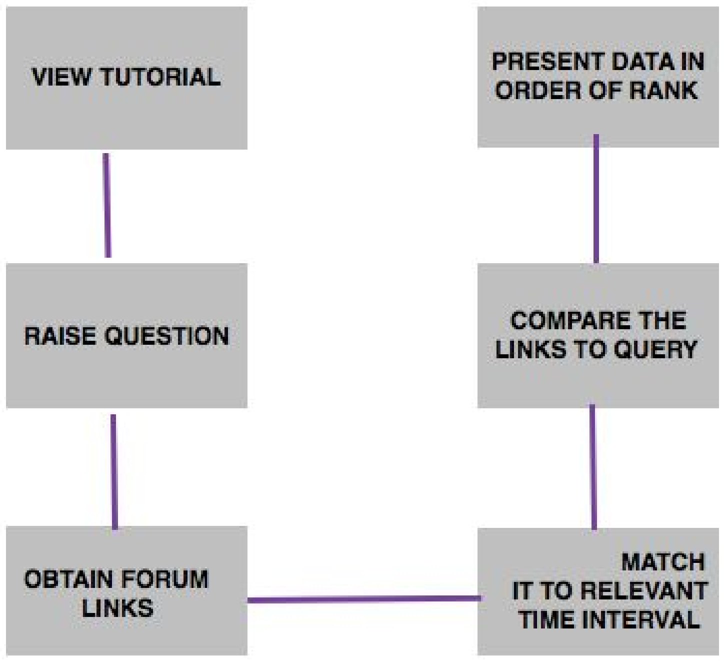

VIEW TUTORIAL
RAISE QUESTION
OBTAIN FORUM LINKS
MATCH IT TO RELEVANT TIME INTERVAL
COMPARE THE LINKS TO QUERY
PRESENT DATA IN ORDER OF RANK

Fig 3.1 High Level Flow Diagram

3.2 ENVIRONMENTS USED IN THE PROJECT

3.2.1 Hardware Requirements

This project mostly includes numerous similarity algorithm using which a database is created. This database is then used to display the links to the user. Therefore, the hardware requirements are minimal as the processes are not too processor / memory intensive.

Basic hardware requirements:

- Processor: Intel Core i3
- RAM: 1 GB
- Hard Disk: 10 GB

3.2.2 Software Requirements

- Language
  - Python
- Technologies
  - Natural Language Toolkit (nltk)
  - word2vec (Gensim)
  - numpy
  - scipy
  - sklearn
  - Matplotlib
- Softwares
  - Browser: Latest version of Firefox

3.2.3 Requirements for the Project

FUNCTIONAL REQUIREMENTS

FR1	Extraction of keywords from transcript snippet and forum link	**Input**: text from snippet and forum **Output:** keywords from the respective text inputs
FR2	Compute similarity	**Input:** keywords/text from snippets and forums **Output:** Relevance tag according to similarity value
FR3	Rank the forums based on their similarity to the video	**Input:** forum links **Output:** ranked list of forums

Table 3.1 *Functional Requirements of Project*

NON-FUNCTIONAL REQUIREMENTS

NF1	Reliability	The system should present correct, precise and useful forum links to the user
NF2	Usability	The system should have a simple UI which is easy to understand and the user must be able to adapt to the same
NF3	Testability	The system should be testable according to the various parameters
NF4	Performance	An essential requirement since the displayed forum links are supposed to change continuously as the video progresses; any delay would mean that the topic being covered in the video and the annotated forums are not in sync
NF5	Availability	The application should be able to recommend forum links 24×7 whenever a user is watching a video that has been annotated
NF6	Network Requirement	No network requirement as the application is not real time, and thus should not require internet. Internet is required to watch the videos, nonetheless.

Table 3.2 *Non-Functional Requirements of Project*

3.2.4 Constraints and Dependencies

- Requires a large set of manually tagged data to develop the model that understands related word
- A huge list of keywords which covers the programming parlance, eg: *array, pointer, static, main,* etc.
- At the time of writing this report, the forum links are static. A dynamic set of forums will require internet.
- Implementation constraints:
    - Programming language:
        - Backend: Python
        - Server: Flask (Python based)
    - Code should be modular, reusable and extendable
- The algorithms are designed to be fast, the links are prefetched to reduce the lag in displaying links to be negligible

3.2.5 Assumptions

- The dataset has been correctly tagged into the three classes: *relevant, slightly relevant,* and *irrelevant*
- All the pickled and database files are present on the server at the time the links are being fetched on the server side
- Python 3 is installed on the server with all the modules mentioned in `requirements.txt`
- On the client side, a browser is required to view the videos and the corresponding links wherein the user makes a selection of the course and therein a topic under that course

3.2.6 Use case diagram for the Requirements

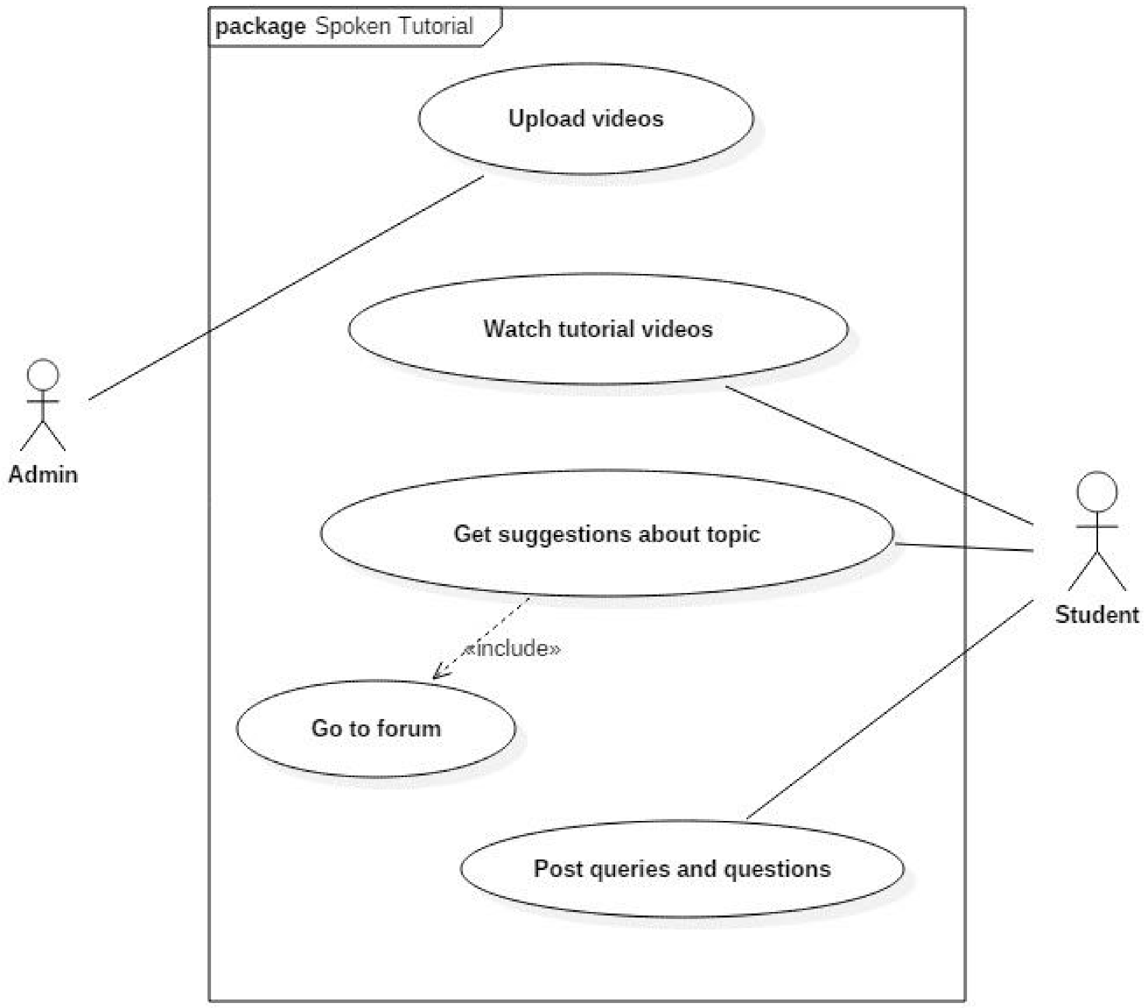


Fig. 3.2 *Use case Diagram for Product Requirement Specifications*

3.2.7 Requirements Traceability Matrix

Testing details

TEST CASE	TEST CASE REQUIREMENT
T1	Important phrases highlighting a concept should be detected as key phrases
T2	The title of a page should be detected as a key phrase
T3	The input text file should be preprocessed appropriately
T4	Check if the results match the timed interval
T5	Check if the tag assigned by algorithm matches the tag manually tagged
T6	Manual checking of the relevance of the top ranked result

Table 3.3 *RTM for Test Case Requirements*

Testing Functional Requirements

FUNCTIONAL REQUIREMENT	TEST CASE
FR1	T1
FR1	T2
FR1	T3
FR2	T3
FR2	T4
FR2	T5
FR3	T4
FR3	T6

Table 3.4 *Testing Functional Requirements*

Requirements Traceability Matrix

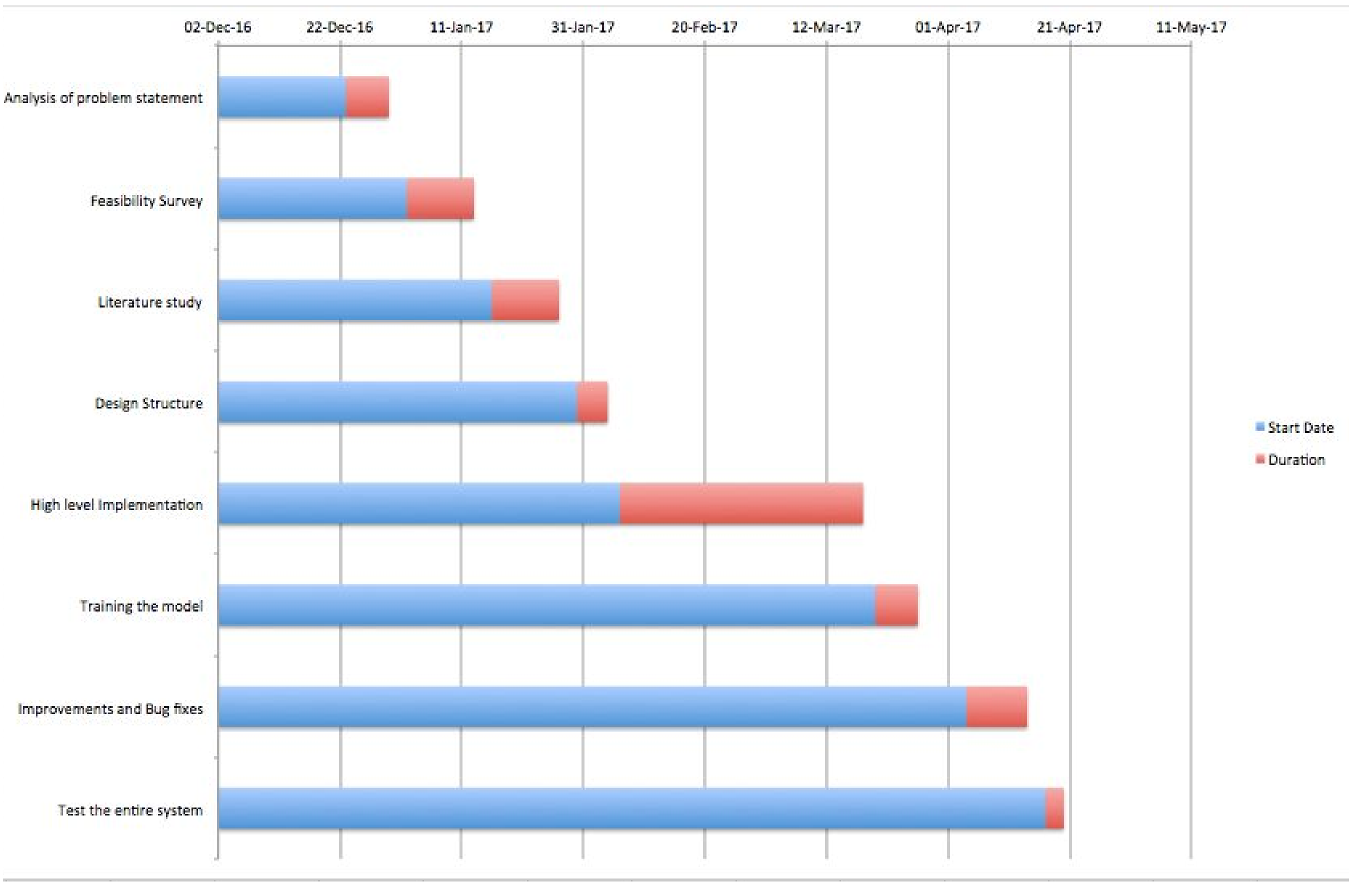
Req ID	Requirement	Design	Implementation	Testing
FR1	Extraction of keywords from transcript snippet and forum link			
FR2	Compute similarity			
FR3	Rank the forums based on their similarity to the video			

Table 3.5 *Updated RTM after Software Requirements Specifications*

4. SCHEDULE

TASK	START DATE	DURATION	END DATE
Analysis of Problem Statement	23/12/2016	7	30/12/2016
Feasibility Study	02/01/2017	11	13/01/2017
Literature Survey	02/01/2017	11	27/01/2017
Design Structure	30/01/2017	5	04/02/2017
Implementation	06/02/2017	40	18/03/2017
Training the Model	20/03/2017	7	03/04/2017
Bug Fixes and Improvements	04/04/2017	10	14/04/2017
Test the entire system	17/04/2017	3	20/04/2017

Table 4.1 *Project Schedule*



GANTT CHART

Fig. 4.1 *Gantt Chart for Project Schedule*

5. SYSTEM DESIGN

5.1 ARCHITECTURE DIAGRAM

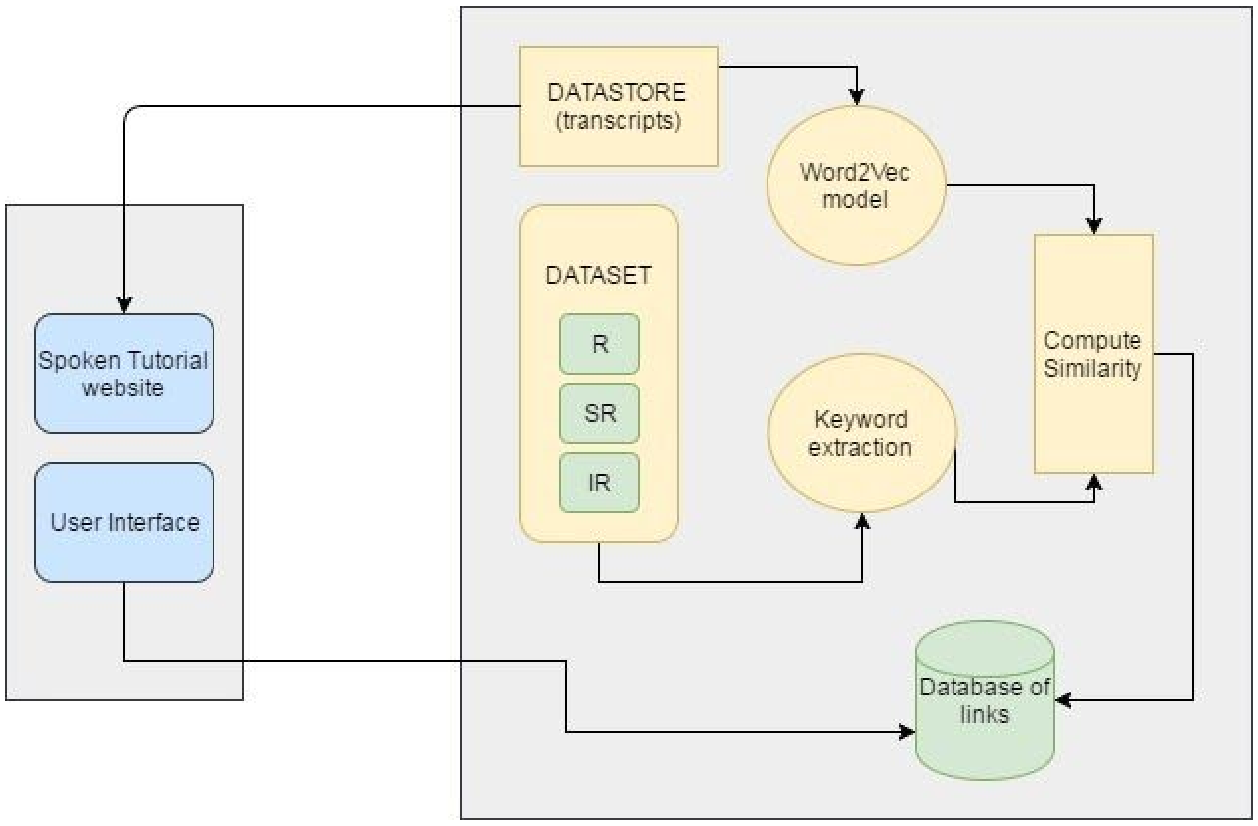


Fig. 5.1 *Architecture Diagram for Project*

Server Side

- Datastore:
    - Videos for which forum links have been updated in the DB
    - Stores transcripts for each video on the Spoken Tutorial website
- Dataset:
    - Stores snippets from each transcript file
    - For each of the above snippets, forum texts are tagged along with relevance
        - Forum text includes question or one of the answers
- Word2Vec model:
    - On each of the transcript files, word2vec is applied to get word embeddings for each word in the transcript
- Keyword extraction:
    - Keywords are extracted from the forum text
    - Keywords are extracted from transcript snippet text
    - Both sets are compared to obtain a measure of similarity
- Hybrid model (compute similarity):
    - A hybrid model which involves an amalgam of algorithms is used to compute the similarity between the snippet text and the forum text in the dataset
- Database:
    - It stores transcript snippet related information:
        - Transcript filename
        - Timestamp
        - Forum links associated with the snippet in order of relevance
        - Relevance

Client Side

- Dropdown menu to choose a topic and corresponding videos
- Video player which plays the chosen video
- A container that displays the top forum links matched for the current time stamp of the video (this list changes dynamically as the video progresses)

5.2 SEQUENCE DIAGRAM

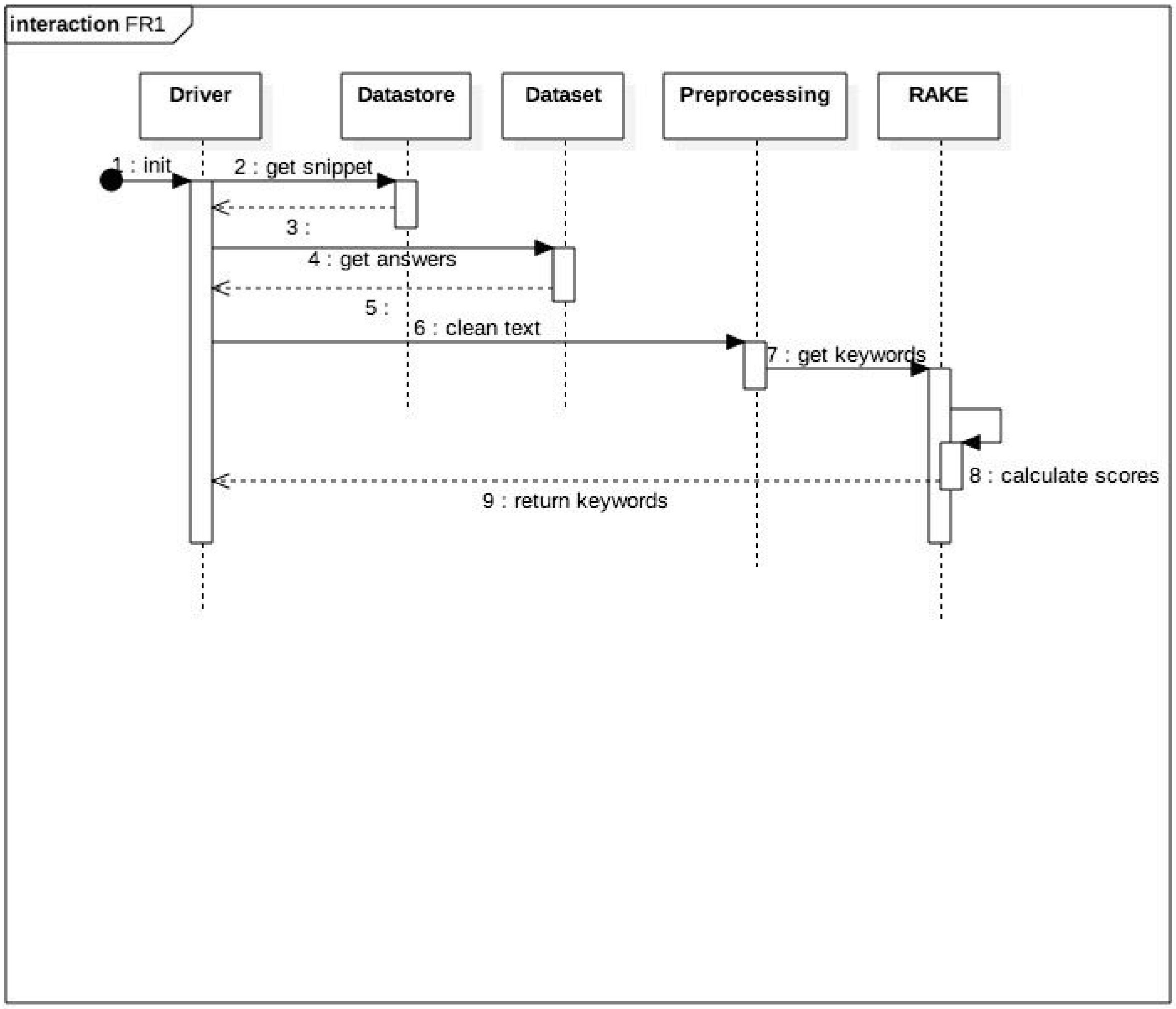


Fig. 5.2 *Sequence Diagram for Keyword Extraction*

Fig. 5.2: Sequence diagram to describe the interaction of components involved in extracting keywords from the transcript snippet and forum text.

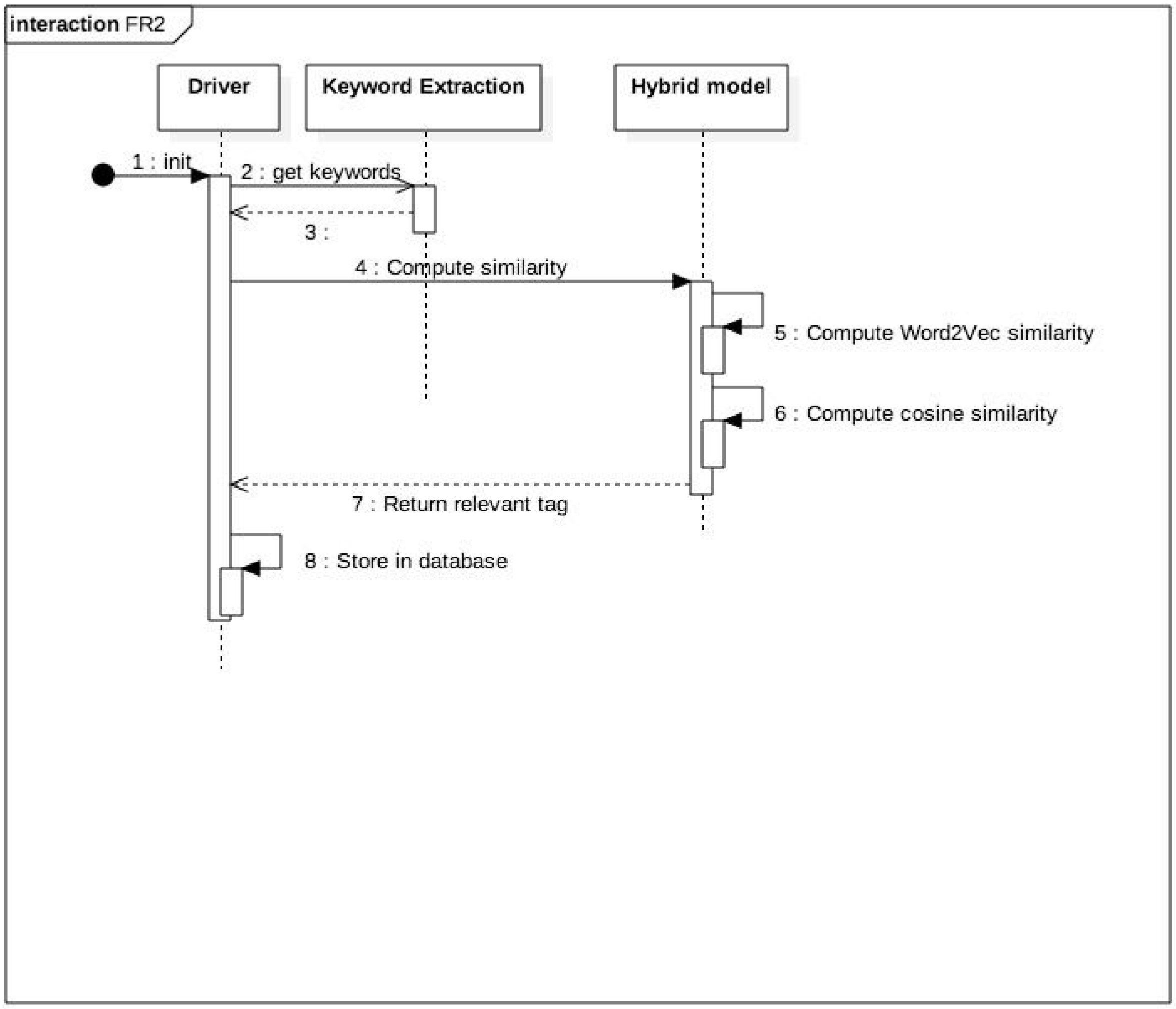


Fig. 5.3 *Sequence Diagram for Similarity Computation*

Fig 5.3: Sequence diagram depicting interaction of components required to compute similarity between transcript snippet and forum text and return the appropriate tag.

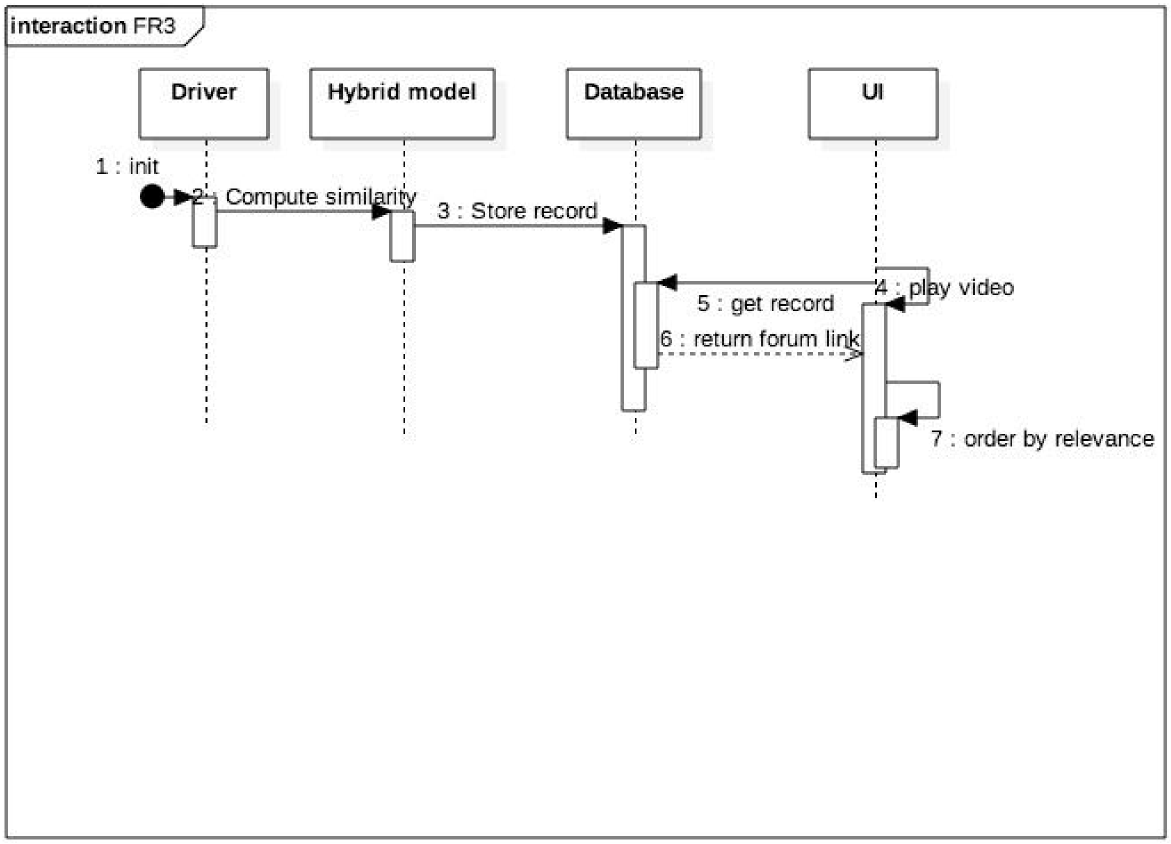


Fig. 5.4 *Sequence Diagram for Links Retrieval*

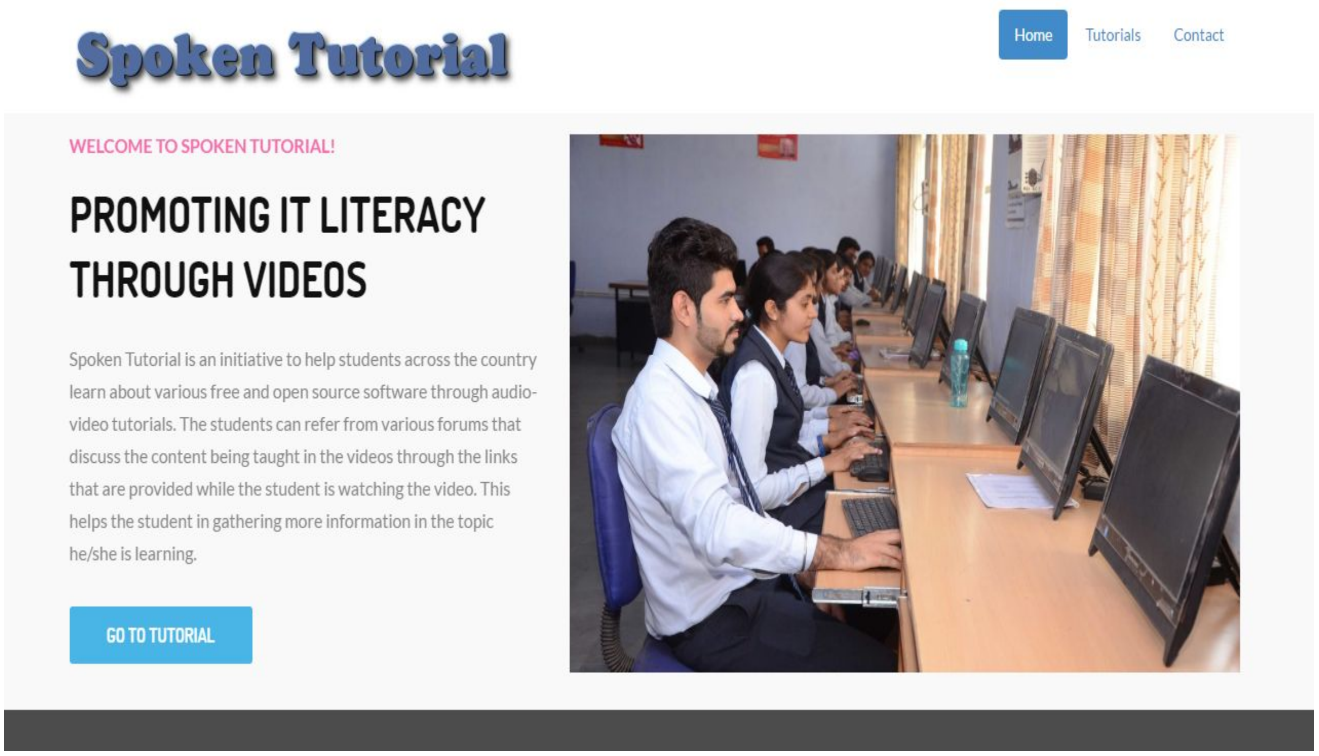
Fig. 5.4: Sequence diagram depicting component interactions to display relevant forum links to the user in accordance to the video time interval.

5.3 UI DESIGN

- A simple web application was developed using the following templates
  - HTML5
  - CSS3
  - Flask (for server)
  - Use of web standards

Homescreen

This is the home page that a student will land when he opens this web page. It gives a brief description about our web page and provides buttons to navigate to other pages.



WELCOME TO SPOKEN TUTORIAL!

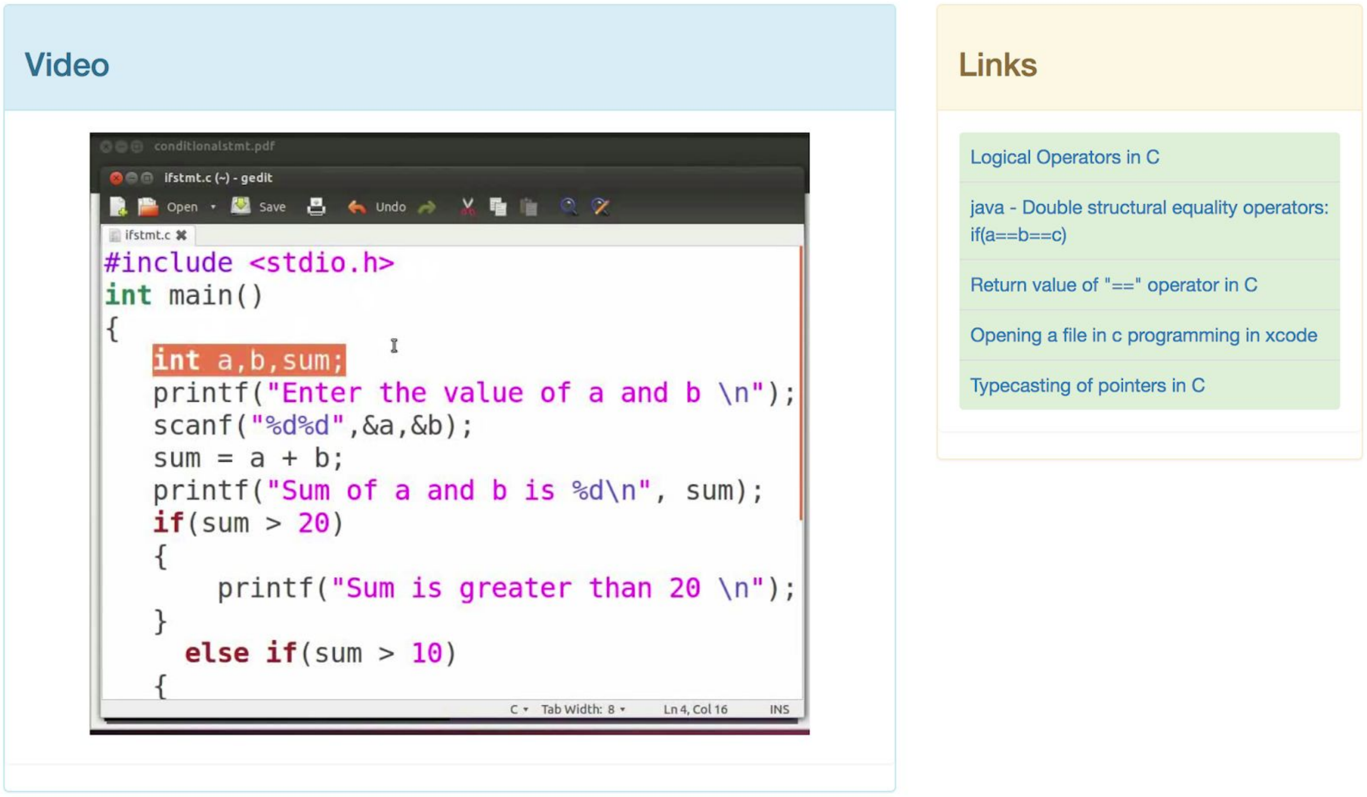
PROMOTING IT LITERACY THROUGH VIDEOS

Spoken Tutorial is an initiative to help students across the country learn about various free and open source software through audio-video tutorials. The students can refer from various forums that discuss the content being taught in the videos through the links that are provided while the student is watching the video. This helps the student in gathering more information in the topic he/she is learning.

Fig. 5.5 *Front Page of UI*

Tutorial Search Page

- In this page, the user can choose a course and a topic in that course from the two dropdowns that are provided.
    - The course dropdown contains a list of available courses like BASH, Advance C, etc.
    - The topics dropdown contains a list of available topics within the chosen course. For example, if the chosen course is BASH, available topics are Introduction to BASH, Arrays and Functions in BASH, etc.
- For a given pair of course and topic, there's a Spoken Tutorial video available.
- As the video is playing, links related to the topic being discussed are fetched from forums like StackOverflow and displayed beside the video as shown.



```
#include <stdio.h>
int main()
{
   int a,b,sum;
   printf("Enter the value of a and b \n");
   scanf("%d%d",&a,&b);
   sum = a + b;
   printf("Sum of a and b is %d\n", sum);
   if(sum > 20)
   {
       printf("Sum is greater than 20 \n");
   }
     else if(sum > 10)
   {
```

- Logical Operators in C
- java - Double structural equality operators: if(a==b==c)
- Return value of "==" operator in C
- Opening a file in c programming in xcode
- Typecasting of pointers in C

Fig. 3.6 *Tutorial Search Page of UI*

5.4 UPDATED RTM

Req ID	**Requirement**	**Architecture Component**	**Implementation**	**Testing**
FR1	Extraction of keywords from transcript snippet and forum link	Keyword Extraction (RAKE)		
FR2	Compute similarity	Hybrid model		
FR3	Rank the forums based on their similarity to the video	User Interface		

Table 5.1 *Updated RTM after System Design*

6. DETAILED DESIGN

6.1. DATA FLOW DIAGRAM

Level 0 Data Flow Diagram

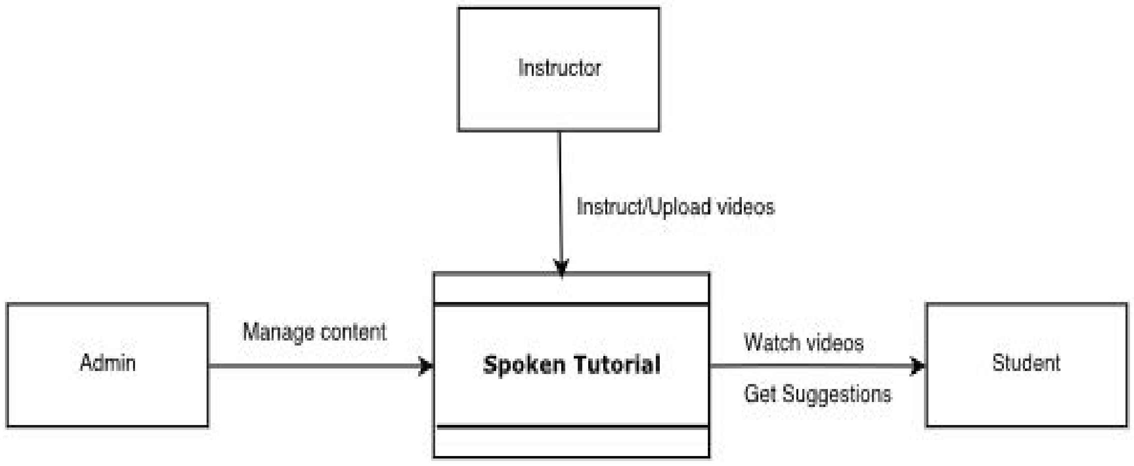


Fig. 6.1 *Level 0 Data Flow Diagram*

- The instructor records videos of him/her lecturing and uploads it
- Student can watch these videos and take down notes
- Admin maintains the backend - maintain a repository of videos and their relevant forum links

Level 1 Data Flow Diagram

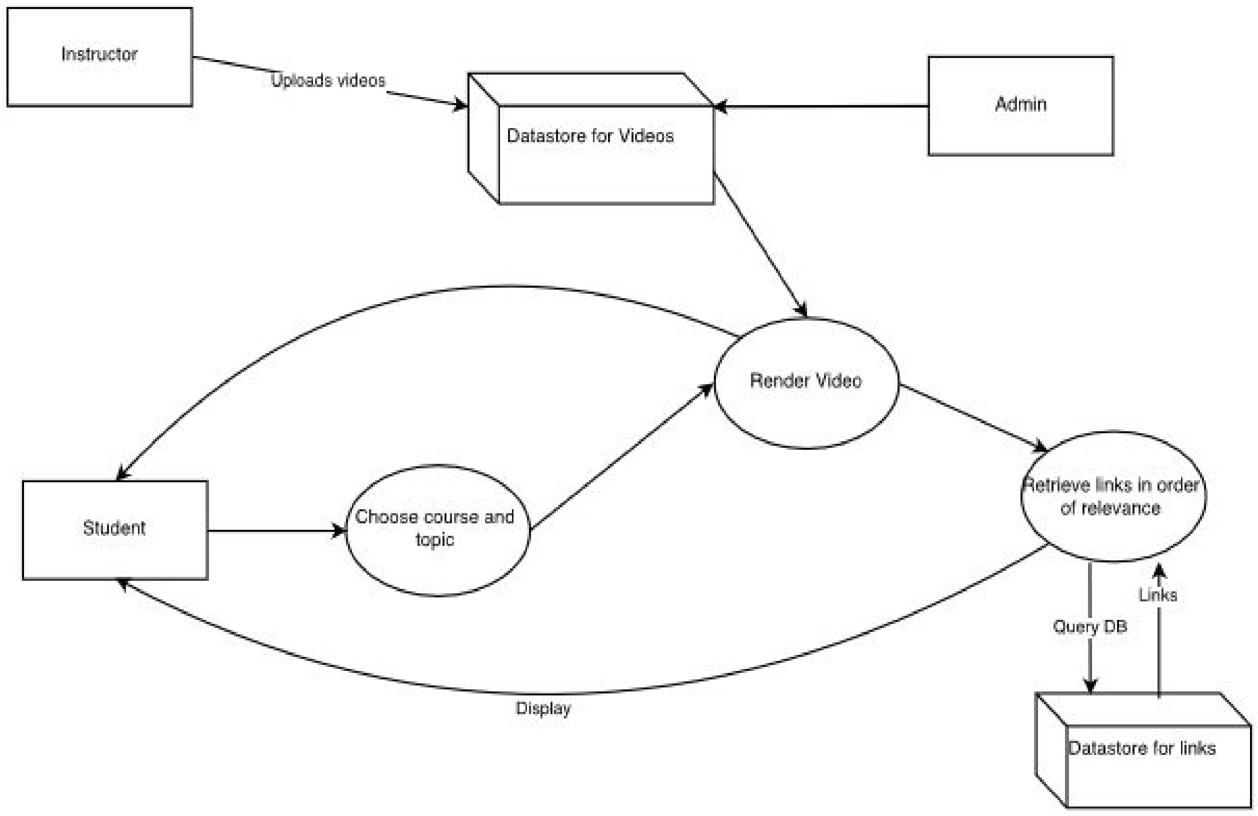


Fig. 6.2 *Level 1 Data Flow Diagram*

- Student can choose course and topic from the options given on the UI
    - Depending on them , the corresponding video is rendered
- Then the links database is queried for links relevant to the rendered video
- The videos database and links database is maintained by admin

Level 2 Data Flow Diagram

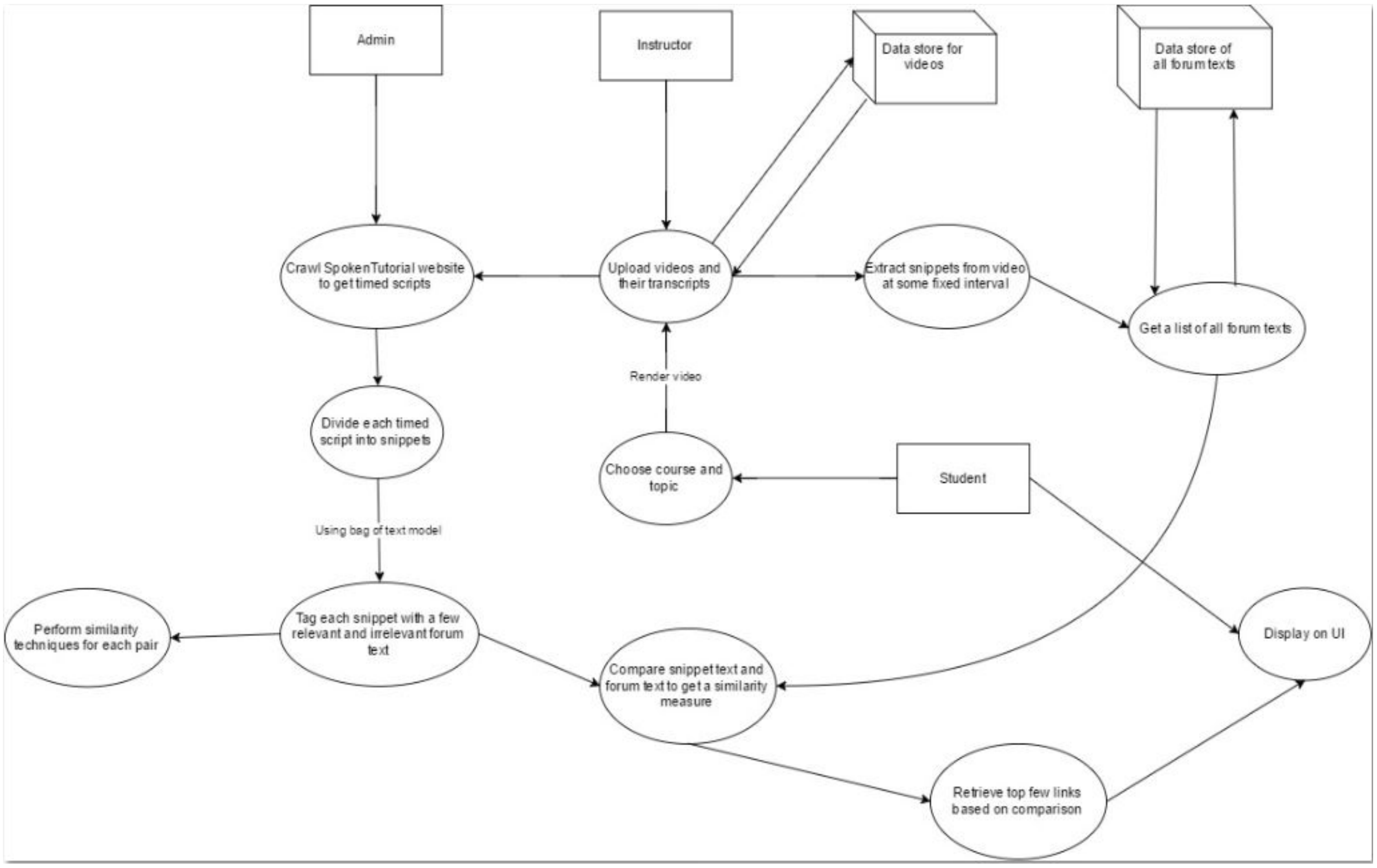

Admin
Instructor
Data store for videos
Data store of all forum texts
Crawl SpokenTutorial website to get timed scripts
Upload videos and their transcripts
Extract snippets from video at some fixed interval
Get a list of all forum texts
Render video
Divide each timed script into snippets
Choose course and topic
Student
Using bag of text model
Perform similarity techniques for each pair
Tag each snippet with a few relevant and irrelevant forum text
Compare snippet text and forum text to get a similarity measure
Display on UI
Retrieve top few links based on comparison

Fig. 6.3 *Level 2 Data Flow Diagram*

Data collection and tagging

- Admin first crawls the Spoken Tutorial website to get all transcripts of all videos and stores them in a datastore
- For the data collection part, each transcript is divided into snippets and manually tagged with forum links
    - Here, we use bag of posts model. Each forum post is considered as a bag of posts wherein the question and each of the answers are compared separately with the snippet under consideration (pairwise similarity) to calculate similarity between a forum post and snippet
    - Each snippet-forum pair is categorized as relevant, irrelevant or slightly relevant
- Similarity between the snippet and forum post is calculated using a hybrid approach of various similarity techniques - keyword extraction followed by cosine similarity, Word2vec similarity and max cosine similarity
    - This file-snippet-forum-similarity is stored as a row in the database

User Interface

- The UI is developed using Python Flask as backend and Javascript and Bootstrap in the front end
    - Python provides a no. of libraries and many plugins with Flask to work with
    - Flask has simple syntax and is a good choice for unit testing of GUI components
- When user navigates to the page, he/she is shown two dropdowns. The first one to choose a course from and the second one to choose topic from (topic list depends on the course choice)
- Depending on the course-topic chosen, the corresponding video is rendered

Retrieving links

- While the student is watching the video, snippets from video are extracted at regular intervals (20 seconds)
    - The links window is refreshed every 20 seconds
- Each snippet is compared against the datastore of forum links
    - Comparison is done using the hybrid approach as described above
- Top forum posts (most relevant to the snippet) are retrieved and displayed beside the playing video in the order of relevance
    - These posts are also stored in a database for future reference (also to save time)
- The student can click on any of these links when he wants more information on the topic he/she is watching

6.2. UPDATED RTM

Req ID	**Requirement**	**Architecture Component**	**Implementation**	**Testing**
FR1	Extraction of keywords from transcript snippet and forum link	Keyword Extraction (RAKE)	`rake.py`	
FR2	Compute similarity	Hybrid model	`hybrid.py`	
FR3	Rank the forums based on their similarity to the video	User Interface	`app.py` `index.html` `tutorial.html`	

Table 6.1 *Updated RTM after Detailed Design*

7. IMPLEMENTATION

7.1 PSEUDO CODE/ALGORITHMS

The aim of this project is to come up with an algorithm that is able to pick out the relevant forum threads by measuring the similarity between a snippet from the transcript and the posts in different threads. There were various algorithms which we benchmarked. Some of them are discussed below.

We started out with a pairwise similarity model and then tweaked the actual cosine similarity algorithm to suit this model. This modified cosine similarity is sampled from *sum cosine similarity* and *max cosine similarity.* The former one does a pairwise cosine similarity between the transcript snippet and all the posts in the forum and then sums up all of them. The final result is calculated by normalizing the sum. Another one is *max cosine similarity* which instead of summing the results, takes the max.

```
def calculate_keyword_similarity(script, forum):

        script_keywords = extract_keywords(script)

        forum_keywords = extract_keywords(forum)

        documents = [script_keywords, forum_keywords]

        similarity = modified_cosine_similarity(documents)

        return similarity
```

Another similarity technique used is *word2vec.* This uses Python's gensim module and similarly a pairwise similarity is calculated and the max of all of them is taken as the final result. The *word2vec* model is created from the transcripts scraped from Spoken Tutorial's website with the following parameters.

```
model = Word2Vec(sentences, size=100, alpha=0.025, window=6, min_count=5,
max_vocab_size=None, iter=100)
```

```
def pairwise_word2vec(script, forum):

        sum_word2vec, max_word2vec = 0, -1

        l = len(forum)

        for post in forum:

                sentences = [script, post]

                sim = calculate_word2vec_similarity(sentences)

                sum_word2vec += sim

                if sim > max_word2vec:

                        max_word2vec = sim

        sum_word2vec /= l

        return sum_word2vec, max_word2vec
```

One challenge while using *word2vec* similarity model was that the default gensim's module can only be used to calculate similarity between two words. Thus, to calculate sentence similarity, the average feature vector of the sentences in question was calculated and the vectors were compared against each other. The calculation of average vectors was performed as follows:

```
def avg_feature_vector(words, model, index2word_set, num_features=100):

        # function to average all words vectors in a given paragraph

        feature_vec = np.zeros((num_features,))

        nwords = 0

        for word in words:

                if word in index2word_set:

                        nwords += 1
                feature_vec = np.add(feature_vec, model[word])
    if nwords > 0:
        feature_vec = np.divide(feature_vec, nwords)
    return feature_vec
```

The most promising algorithms among these were then voted to create a hybrid algorithm which classifies a forum thread into one of the three classes: *relevant, slightly relevant, irrelevant.*

```
def hybrid_similarity(script, forum):
    similarity = calculate_keyword_similarity(script, forum)
    if similarity < low:
        relevance = 'IR'
    else:
        # max word2vec
        similarity = pairwise_word2vec(script, forum)[1]
        if similarity > high:
            relevance = 'R'
        else:
            relevance = 'SR'
    return relevance
```

7.2 CODEBASE STRUCTURE

- Core Logic Algorithms
    - `similarity.py`
    - `hybrid.py`
    - rake
        - `smart_stoplist.txt`
        - `rake.py`
- Processor
    - `database.py`
    - `preprocess.py`
    - `scrape.py`
    - `video_fetch.py`
    - `window.py`
    - `word2vec_transcripts.py`
    - `title.py`
- Data
    - data
        - Advance C
        - C and Cpp
        - Bash
        - links_2.2.csv
- User Interface
    - UI
        - `app.py`
        - `index.html`
        - `Tutorial.html`

The codebase is divided into four parts as structured above.

- Core Logic Algorithms - The codes classified here comprises the approaches which are central to the project and produce end results as required.
- Processor - The codes in this category perform preprocessing functionalities on the data which will be used by the codes under ‘core logic algorithms’.
- Data - These are static files and folders which consist of the data required for use.
- User Interface - These files consist of the html pages served to the front end and the python flask application which starts up the server (localhost).

Github is used as the central repository to maintain version control and manage source code .

7.3 CODING GUIDELINES

This section describes the coding guidelines used throughout our project. Majority of the project has been coded in Python and we have used **PEP8** style guide consistently as the coding convention. We have made some modifications to the convention for our convenience as well which has been mentioned below.

7.3.1 Code Layout

- Indentation
    - Use 1 tab per indentation level for convenience (fewer keystrokes)
    - Mixed indentation is not allowed and is enforced by default in Python 3
- Blank lines
    - Top-level function and class definitions are surrounded with two blank lines
    - Method definitions inside a class are surrounded by a single blank line
    - Blank lines are used in functions, sparingly, to indicate logical sections
- File encoding
    - Code in the core Python distribution always use UTF-8
    - Encoding is enforced for files being opened even though Python 3 defaults to UTF-8
    - All identifiers use English words as much as possible
    - String literals and comments use only ASCII characters
- Imports
    - Imports from separate modules are done on separate lines
    - If importing from the same module, it can be done on the same line
    - Imports are put at the top of the file
    - Imports are done in the following order:
        - Standard library
        - Third party library
        - Local application modules

- String quotes
  - Single quotes are used to declare strings
  - If the string contains a single quote character (`'`), it is enclosed in double quotes
- Whitespace
  - Single space character after comma (`, `)
  - Binary operators are surrounded with single space characters (` + `)
  - Extraneous whitespace characters are avoided as much as possible

7.3.2 Naming conventions

- Naming styles
  - Variables have all lowercase characters (`lowercase`)
  - Words are separated using underscores (`lowercase_with_underscores`)
  - To avoid conflict with Python keywords, variable names with a trailing underscore is used (`max_`). However, a different name is preferred.
  - Constants are written in full caps (`CONSTANT`)
  - Filenames also follow the same convention (`file_name`)
  - Class names follow upper camelcase (`ClassName`)

7.3.3 Programming conventions

- Chained comparisons are simplified (`x > 0 and x < 1` is replaced with `0 < x < 1`)
- Comparisons to singletons like `None` is never done; among the below three ways only the first is preferred, while the second one is also acceptable (equality operators should NOT be used)
  - `if x:`
  - `if x is None:`
  - `if x == None:`
- Exceptions are derived from `Exception` and not `BaseException`

7.3.4 HTML conventions

- Element and attribute names are lowercase

```
<section>
        <p>This is a paragraph.</p>
</section>
```

- Non empty elements are closed with the corresponding tags

```
<p>This is a paragraph.</p>
```

- Empty elements are closed with short tags

```
<br />
```

- Attribute names are always double quoted

```
<table class="striped">
```

- An id is used to link an anchor inside an HTML document

```
<a href="#section">link</a>
<div id="section"></div>
```

- Only starting comments are used. Comments are inserted in this format:

```
<!-- / name-of-class-or-id →
```

7.3.5 CSS conventions

- Tabs are used for indenting the properties
- One blank line is added between blocks
- Each selector should be on its own line, ending in either a comma or an opening curly brace.
- Property-value pairs should be on their own line, with one tab of indentation and an ending semicolon.
- The closing brace should be flush left, using the same level of indentation as the opening selector.

7.4 SAMPLE CODE

Preprocessing

```
def clean(text):
        text = text.lower()
        text = re.sub('[#.,$%|~\-/&\"\'`*+=!?;(){}^]', '', text)
        text = re.sub('\n ', '\n', text)
        text = re.sub('\n+', '\n', text)
        text = re.sub(' +', ' ', text)
        matches = re.findall(r'(\d+:\d+)[ \n]+(((?!\d+:\d+).)*)', text, re.S)

        cleaned_lines = []
        for match in matches:
                timestamp = match[0]
                line = re.sub('\n+', ' ', match[1])

                if 'end of this tutorial' in line:
                        break

                line = line.strip()
                line = re.sub(r':([ ]?)', r'\1', line)
                words = line.split(' ')
                filtered_words = [word for word in words if word not in stop_words]
                cleaned_lines.append((' '.join(filtered_words), timestamp))

        return cleaned_lines
```

Word2Vec similarity

```
def calculate_word2vec_similarity(sentences):

        model = Word2Vec.load('data/snippet.model')

        word_index_set = set(model.index2word)

        script_vector = avg_feature_vector(sentences[0].split(), model, word_index_set)

        forum_vector = avg_feature_vector(sentences[1].split(), model, word_index_set)

        word2vec_similarity = 1 - cosine(script_vector, forum_vector)

        return word2vec_similarity
```

Cosine similarity using pairwise model

```
def get_pairwise_similarity(snippet, forum_texts):

        sum_sim, sub_sim, max_sim, min_sim = 0, 1, 0, 999

        l = len(forum_texts)

        for post in forum_texts:

                temp = calculate_similarity([snippet, post])[0][1]

                # sum the pairwise similarity for each post (normalize in the end)

                sum_sim += temp

                # take the max of pairwise similarity for the posts

                if temp > max_sim:

                        max_sim = temp

                # take the min of pairwise similarity for the posts

                if temp < min_sim:

                        min_sim = temp

        sum_sim /= l

        sub_sim -= sum_sim

        return [sum_sim, sub_sim, max_sim, min_sim]
```

Average feature vector for comparing sentences using word2vec

```
def avg_feature_vector(words, model, index2word_set, num_features=100):

        # function to average all words vectors in a given paragraph

        feature_vec = np.zeros((num_features,), dtype="float32")

        nwords = 0


        for word in words:

                if word in index2word_set:

                        nwords += 1

                        feature_vec = np.add(feature_vec, model[word])
        if nwords > 0:
                feature_vec = np.divide(feature_vec, nwords)
        return feature_vec
```

Hybrid similarity approach

```
def hybrid_similarity(script, forum):
        # comparison param
        high = 0.5
        low = 0.1
        sentences = [script, ' '.join(forum)]
        similarity = calculate_keyword_similarity(sentences)
        if similarity < low:
                relevance = 'IR'
        else:
                # max word2vec
                similarity = pairwise_word2vec(script, forum)[1]
                if similarity > high:
                        relevance = 'R'
                else:
                        relevance = 'SR'
        return relevance, similarity
```

7.5 UNIT TEST CASES

Function	Test Case ID	Test case input	Expected result	Outcome
`rake.py`	T1	Single line of definition (snippet)	words defining the concept are top scoring	PASS
`rake.py`	T2	question of a forum link	title as highest scoring phrase	PASS
`preprocess.py`	T3	Text snippet from transcript or forum text	All stopwords are removed	PASS
`preprocess.py`	T3	Text snippet from transcript or forum text	All special characters are removed	PASS
`preprocess.py`	T3	Text snippet from transcript or forum text	Extra whitespaces are removed	PASS
`word2vec_transcripts.py`	T5	Irrelevant data	Low similarity value	PASS [Fixed]
`word2vec_transcripts.py`	T5	Relevant data	High similarity value	PASS

`index.html`	T6	Video with transcripts	Relevant links show up	PASS
`index.html`	T6	Video with transcripts	Clickable titles	PASS[Fixed]
`index.html`	T4	Video with transcripts	Forum links at 20s intervals	PASS

Table 7.1 *Unit Test Cases*

7.6 METRICS FOR UNIT TEST CASES

- Average number of test cases per requirement : 2
- Amount of data available for testing : 216 relevant links, 46 irrelevant along with their corresponding transcript snippets
- Pending defects : 0
- Defects detected: 2
- Severity of defects: Moderate
- Amount of data covered during execution: 100%

7.7 UPDATED RTM

Req ID	Requirement	Architecture Component	Implementation	Unit Test
FR1	Extraction of keywords from transcript snippet and forum link	Keyword Extraction (RAKE)	`rake.py`	`preprocess.py` `rake.py`
FR2	Compute similarity	Hybrid model	`hybrid.py`	`word2vec_transcripts.py`
FR3	Rank the forums based on their similarity to the video	User Interface	`app.py` `index.html` `tutorial.html`	`index.html`

Table 7.2 *Updated RTM after Unit Testing*

8. TESTING

8.1. SYSTEM/FUNCTION TEST SPECIFICATION

Test case specification	Expected result	Outcome
Hardware	Application should not consume too much resources	PASS[fixed]
Usability	Ease of use while navigating through the website	PASS
Performance	Time taken to display links should be optimal	PASS
Portability	The application should run on different OS versions	PASS
Accuracy	The selected method should produce results close to baseline accuracy	PASS
Regression	New patches should not break the code	PASS

Table. 8.1 *Test Specifications for Product*

8.2. TEST ENVIRONMENT USED

8.2.1 Operating System

- Linux (Ubuntu)
- Mac OS X
- Windows

8.2.2 Hardware Specifications

- Memory : 4GB / 8GB

8.2.3 Server

- Apache
- Flask

8.2.4 Browser

- Firefox
- Chrome

8.2.5 Test Data

- Text from transcripts
- Text from forum links
- Spoken tutorial videos

8.3. TEST PROCEDURE

- Each function which will be used to compute the similarity measure is individually tested against the dataset.
- Multiple methods as described in the following section were also run against this dataset and benchmarked to pick the algorithm most suitable with the data in hand.
- The whole application was tested on a web page hosted by flask
- The frequency and relevance of the links was tested.
- Various non functional requirements were evaluated by monitoring hardware resources and tweaking UI design.

8.4. EXAMPLE TEST RESULT

- User Interface
  - When the video begins to play forum links appear on the side every 20s
  - The links displayed do show certain discrepancies regarding its relevance however 80% of the displayed links are relevant to the topic being discussed.
- Similarity computation
  - The hybrid model takes into picture the word2Vec as well as keyword cosine and the best mix is fixated as the model.
  - Given the size of data available the accuracy obtained is a satisfactory number.
- Keyword Computation
  - Question and its answers were extracted from a stackoverflow link and given as input separately to rake.py
  - It was observed that the title gave the highest score which is the expected behavior.
  - Since this approach is formula based certain words with no relevance are also detected as keywords.

8.5. TEST METRICS

- Baseline accuracy : 60%
- Percentage of relevant links displayed : 80%
- Defects corrected : 3

8.6. UPDATED RTM

Req ID	Requirement	Architecture Component	Implementation	Unit Test	System Test
FR1	Extraction of keywords from transcript snippet and forum link	Keyword Extraction (RAKE)	`rake.py`	`preprocess.py` `rake.py`	-
FR2	Compute similarity	Hybrid model	`hybrid.py`	`word2vec_transcripts.py`	`window.py`
FR3	Rank the forums based on their similarity to the video	User Interface	`app.py` `index.html` `tutorial.html`	`index.html`	`app.py` `index.html` `tutorial.html`

Table 8.2 *Updated RTM after System Testing*

9. RESULTS AND DISCUSSIONS

We started out with modified versions of some well known similarity techniques. The results were quite varied. The following graphs cover these similarity algorithms and the results obtained from them:

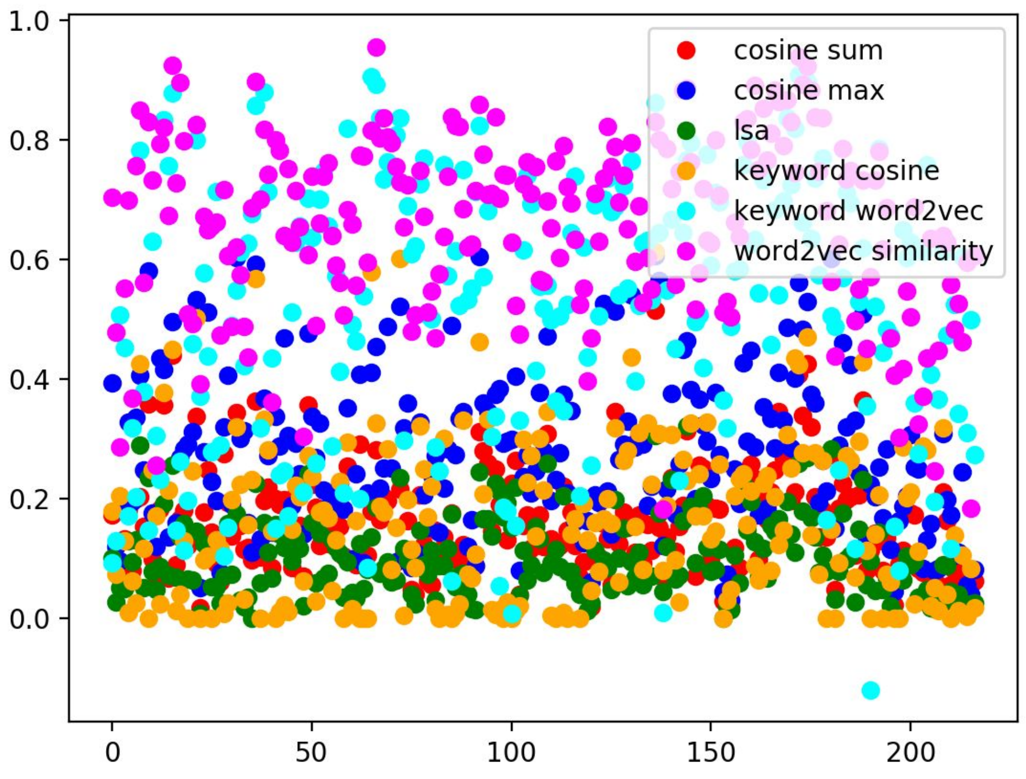


Fig. 9.1 *Output of Different Similarity Techniques*

Fig. 9.1: A total of 217 relevant transcript snippets and a corresponding forum link were compared using these algorithms. *x-axis* is the index of the tuple we are comparing while *y-axis* is the similarity (between 0-1)

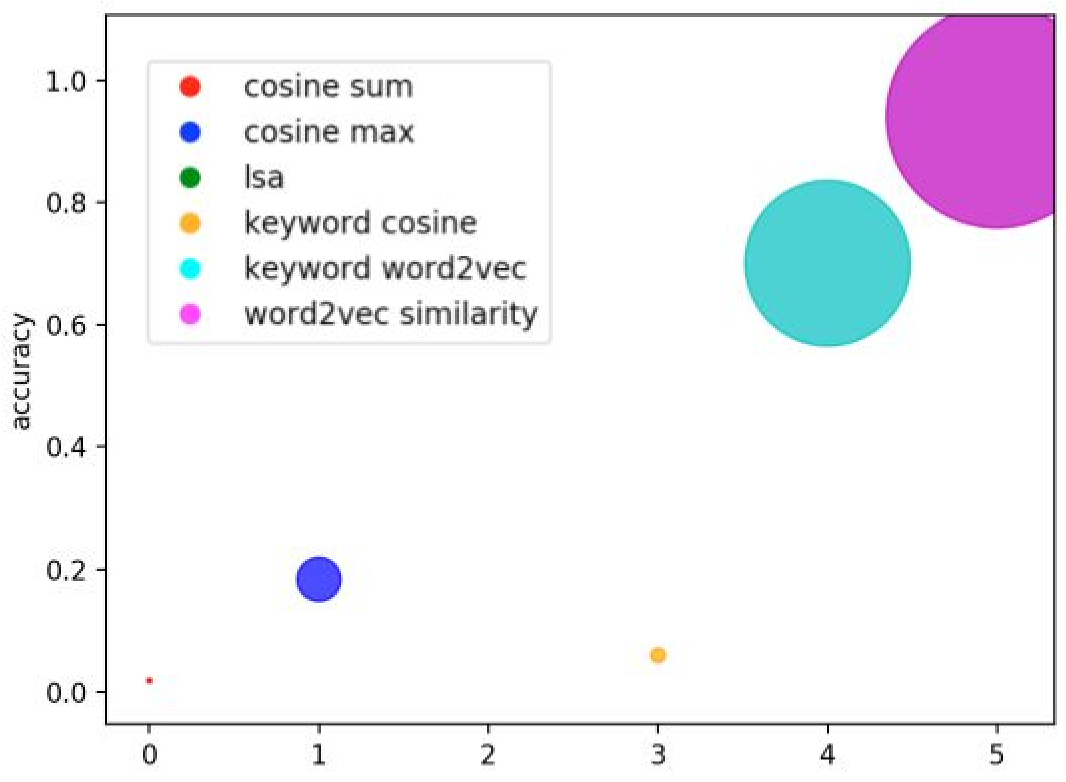


Fig. 9.2 *Accuracy of Different Similarity Techniques*

Fig. 9.2: We have a baseline similarity and if the value of the similarity obtained is greater than this, we consider it a hit otherwise a miss. And with the following equation, we calculate the accuracy:

$$accuracy = hit \ / \ (hit \ + \ miss)$$

We tried out several approaches to calculate the similarity between the transcript snippet and the forum threads. These approaches combined several vanilla algorithms and tried to leverage the fact that different algorithm might be able to find different classes better. The different hybrid approaches were as follows:

Approach I

- Compute the keyword-based cosine similarity
- If it is high (relevant), use word2vec
- If it is low (irrelevant), use keyword-based cosine itself
- Otherwise, (slightly relevant) use max-cosine

Approach II

- Compute keyword cosine
    - If it is < low, irrelevant
- Else, compute word2vec similarity
- If it is high (relevant), it can be R or SR
    - Compute max cosine and see if it lies between (low, high) -> slightly relevant
    - Else, relevant

Approach III

- Compute keyword cosine
    - If < low, then IR
- Else, compute word2vec
    - If > high, then R
    - Else, SR

Approach IV

- Plain word2vec

High	**Low**	**Accuracy (%)**
Approach I		
0.40	0.2	24.0
0.35	0.15	21.18
0.30	0.15	**35.34**
Approach II		
0.50	0.20	20.79
0.40	0.20	**27.53**
Approach III		
0.60	0.15	41.01
0.65	0.10	**65.22**
Approach IV		
0.40	0.20	51.12
0.50	0.30	**55.34**

Table 9.1 *Accuracy of Different Hybrid Approaches*

Inferences

- Regular word2vec *seemingly* gives better accuracy than the hybrid approaches
- This might, in fact, be due to the prominence of relevant results in our database
    - Relevant: 216
    - Slightly relevant: 92
    - Irrelevant: 46
- Other than this, approach III is also performing reasonably well
    - Tweaking the parameters might give even better results in this case
- Guessing an irrelevant result is not much of a problem
- The best approach would be the one which is able to differentiate between **relevant** and **slightly relevant** results

10. RETROSPECTIVE

To build a model a dataset of forum links was needed, which had to be manually searched and tagged (Relevant, Slightly Relevant and Irrelevant)as there was no pre existing data of the type and format required. The initial tutorial videos targeted was Libre Office. However, there was a discrepancy in the level of the tutorial and the the questions on the forum (LibreOffice.org).

The tutorials focus on the very basics of concepts while the questions posted concentrated on the more complex aspects of the topic. Due to this reason the tutorial in focus had to be changed to one where sufficient data was available.

Data size proved to be a drawback through this project. While the topics chosen (Advance C, C and Cpp, Bash) had a good amount of forum links, the small size of each transcript snippet and lack of more forum links created roadblocks in many algorithms which, with a large dataset gave good similarity results between two texts. Semantic similarity techniques and topic modelling algorithms suffered as a direct consequence. After cleaning text the amount of significant words were too less to train the model on.

However, since the data was tagged to three classes, certain algorithms were found to work very well depending on the class. As inferred in section 9, keyword matching gave a high accuracy for irrelevant data as it takes into consideration the common terms. Word2Vec worked well with relevant data and by tweaking parameters and training it it for a good amount of time the results improved. Hence, using these results a hybrid model was created using baseline parameters to build a complete working model to detect similarity.

The project can be enhanced further by utilising deep learning techniques in many areas like keyword matching, method of reducing cosine values to a single term and choosing the best hybrid model as well as utilising features in the forums such as upvotes. A mock user interface has been created which runs on localhost but for real world usage it should be loaded onto IIT Bombay’s server.

In conclusion the smooth and rocky aspects can be summarised as given below.

What went well in the project

- The end goal was successfully met
- A decent accuracy was obtained
- An effective similarity measure using hybrid approaches was developed with good outcomes
- The keyword algorithm was producing good results for the small size of data using the formulaic method
- Video intervals were successfully annotated with relevant forum links

Roadblocks in the project

- Gathering data corresponding to the simplistic level of the topics explained in the tutorials proved to be a challenge.
- Due to the size of the data, many techniques to detect semantic similarity failed to produce significant results
- Topic modelling also proved to be a dead end due to the small size of transcripts.

11. BIBLIOGRAPHY

[1] Gomaa, Fahmy (2013) "*A Survey of Text Similarity Approaches*"

[2] Pattabiraman, Sodhi, Zhai (2013), "*Exploiting Forum Thread Structures to Improve Thread Clustering.*"

[3] Amit, Deepak, Dinesh (2012), " *Retrieving Similar Discussion Forum Threads*."

[4] Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, *Introduction to Information Retrieval*, Cambridge University Press. 2008.

[5] Edwin Chen, "Introduction to Latent Dirichlet Allocation", *http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/*,22 August 2011

[6] Alyona Medelyan "NLP keyword extraction tutorial with RAKE and Maui", *https://www.airpair.com/nlp/keyword-extraction-tutorial*

[7] "Distance and Similarity Measures", *https://reference.wolfram.com/language/guide/DistanceAndSimilarityMeasures.html*

[8] Steven Loria, "Tutorial: What is WordNet? A Conceptual Introduction Using Python", *http://stevenloria.com/tutorial-wordnet-textblob/*, October 26, 2014

12. USER MANUAL

- Install all required python packages
  - `lxml`
  - `nltk`
  - `gensim`
  - `html2text`
  - `py-stackexchange`
  - `numpy`
  - `scipy`
  - `sklearn`
  - `matplotlib`
  - `Flask`
- Run `app.py`
- Open a web browser and type the url `http://localhost:5000`