VISVESVARAYA TECHNOLOGICAL UNIVERSITY,

JNANA SANGAMA, BELGAUM - 590014

A Project Report on

ASSESSMENT OF QUESTION QUALITY USING BLOOM'S TAXONOMY

Submitted in partial fulfilment of the requirements for the award of the degree of

Computer Science & Engineering

Submitted by:

1PI13CS092	Mohit Surana
1PI13CS147	Shiva Karnad Deviah
1PI13CS150	Shrey Agarwal

Under the guidance of:

Prof. Nitin V Pujari
HoD, PESIT - CSE

Jan – May 2017

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

PES INSTITUTE OF TECHNOLOGY,

(AN AUTONOMOUS INSTITUTE UNDER VTU, BELGAUM AND UGC, NEW DELHI)

100FT RING ROAD, BSK 3RD STAGE, BENGALURU - 560085

Assessment of Question Quality using Bloom's Taxonomy

Mohit Surana, Shiva K Deviah, Shrey Agarwal

Under the guidance of Prof. Nitin V Pujari

PES INSTITUTE OF TECHNOLOGY

(An Autonomous Institute under VTU, Belgaum)
100 Feet Ring Road, BSK- III Stage, Bangalore – 560 085

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Certificate

Certified that the eighth semester project work titled "**Assessment of Question Quality using Bloom's Taxonomy"** is a bonafide work carried out by

Mohit Surana	**1PI13CS092**
Shiva Karnad Deviah	**1PI13CS147**
Shrey Agarwal	**1PI13CS150**

in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of Visvesvaraya Technological University, Belgaum during the academic semester January 2017 – May 2017. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said Bachelor of Engineering.

Signature of the Guide	Signature of the HOD	Signature of the Principal,
Prof. Nitin V. Pujari	**Prof. Nitin V. Pujari**	**Dr. K S Sridhar**

External Viva

Name of Examiners	Signature with Date
_____	_____
_____	_____

ABSTRACT

Bloom's Taxonomy has been in existence for over half a century. The main idea behind this model was to promote higher forms of thinking in education, such as analyzing and evaluating concepts, processes, procedures, and principles, rather than simple rote learning. In evaluating the usefulness of Bloom's Taxonomy, only the cognitive domain is usually considered. This is a well-researched topic and much work has been done on the subject. Bloom's Taxonomy has proved to be a good pedagogic tool in the field of education. However, in evaluating information, even the cognitive processes and knowledge level must be considered. This version of the taxonomy considering both the cognitive domain as well as the cognitive processes dimension is called the Bloom's Modified Taxonomy. The aim of this project will be to explore the various techniques in which the modified taxonomic principles can be applied, and to analyse the accuracy of each of these applications.

ACKNOWLEDGEMENT

It gives us great pleasure to acknowledge the support, guidance and motivation rendered by our lecturer and HoD, **Prof. Nitin V Pujari**, as the project guide. We express our profound gratitude for his continuous support and invaluable guidance in turning the project into reality.

We would also like to express our sincere thanks to **Prof. Badri Prasad** for handling all our mundane administrative affairs with cheery aplomb.

We would also like to express our gratitude to **Prof. K S Sridhar**, Principal of our esteemed institution, for giving us the opportunity to work on this project.

We would also like to thank our friends and classmates for taking the time to look at our project and offering their invaluable criticism, all of which resulted in the betterment of the project.

Finally, we would like to thank God for blessing us with the enthusiasm and willpower to strive towards our goal.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1. Introduction

Bloom's Taxonomy, in simple terms, quantifies the level of knowledge and skill required to carry out some task. The original version of Bloom's Taxonomy (Bloom, 1956) [1] documents only the Cognitive, or Skill domain, which consists of the following:

1. Remember
    - Recall memorized information. Different keywords include define, describe, identify, recall, etc.
2. Understand
    - Understanding that enables one to explain in one's own words some problem or how to perform a task. Different keywords include estimate, explain with example, summarise, and interpret.
3. Apply
    - Use a concept to good effect in a new situation; application of knowledge. Keywords include apply, demonstrate, modify, solve, and use.
4. Analyse
    - Decompose a complicated concept into smaller components so that the organizational structure may be understood. Keywords include breakdown, distinguish, identify, illustrate, and infer.
5. Evaluate
    - Judgements or informed opinions about the value of a concept. Keywords include compare, contrast, justify, evaluate, and explain.
6. Create
    - Create a new concept/structure/pattern from existing elements. Keywords include devise, design, modify, plan, reorganise, rewrite, compose, combine, compile, and categorise.

A modified version of Bloom's Taxonomy (Anderson and Krathwohl, 2001) [2] considers this Knowledge domain in detail:

1. Factual
    - The basic level of knowledge that anyone must be acquainted with to solve problems
2. Conceptual
    - The interrelationships among the basic elements; the understanding behind how things work
3. Procedural
    - The knowledge to do something
4. Metacognitive
    - Knowledge of cognition in general as well as awareness and knowledge of one's own cognition

Knowledge Dimension	Cognitive Process Dimension					
	1. Remember	2. Understand	3. Apply	4. Analyze	5. Evaluate	6. Create
Factual Knowledge						
Conceptual Knowledge						
Procedural Knowledge						
Metacognitive Knowledge						

Figure 1-1- Revised version of Bloom's Taxonomy by Anderson and Krathwohl, 2001

Together, these two domains form the Knowledge Matrix of the revised version of Bloom's Taxonomy. Currently, there exists no tool in the open source community that assesses question quality according to Bloom's Modified Matrix, considering both the Cognitive and Knowledge dimensions together. Furthermore, existing work in this domain considers levels to have a flat relationship; rather than hierarchical, which in actuality, they are. For example, in order to explain the rationale behind a technique or concept, you need knowledge and understanding of the problem domain too. Our system will aim to capture the hierarchical

structure of the levels during the process of question classification.

A successful implementation of this system will find many practical uses in the field of education. These include:

1. Assess the quality of lecture delivery; analysing students' doubts after a lecture to determine their understanding of the topic post lecture
2. Automated question paper setting; selecting questions based on their difficulty level.
3. Weighted GPA system; apply weightage to subject grade by analysing question papers set for that subject.

1.1. Problem Definition

This project describes various techniques to assess the difficulty level of academic questions according to the taxonomic principles of the Cognitive as well as the Knowledge domain. For the Knowledge domain, a combination of Naive Bayesian-based document classification, Latent Dirichlet Allocation (LDA), Latent Semantic Analysis (LSA) and Doc2vec cosine similarity is carried out to gauge the "obviousness" of a question. For the cognitive dimension, an ensemble of the Support Vector Machine (SVM), Bidirectional Recurrent Neural Networks (BRNNs) and Multinomial Naïve Bayes Classifier (MNBC) has been built.

This project also explores some practical use cases applications of the system as a pedagogic tool to assist instructors.

The project can be divided into three main phases:

1. Dataset collection

This is a first attempt at assessing questions based on Bloom's Modified Taxonomy. Data has been collected from various resources, these will be mentioned in Section 1.2.

2. Training

Train two classifiers for both the dimensions separately and get a probability distribution for each question. Each classifier takes as input, the question, and provides a probability distribution of the level as output. The outputs from each classifier are combined and

displayed onto a heat map.

3. Evaluation

The third, and most important part of our project is to evaluate the validity of our models, through statistical analysis and human recall.

1.2. Generic Proposed Solution

Our approach to solve the problem at hand will involve the following steps:

I. Dataset Generation

Extraction of keywords from the mini-world reference (textbook/online articles), followed by validation of the keyword by checking the context attached to it (this has been done using DBPedia/Wikipedia). The code for validation will also be useful as we evaluate the relevance of a question according to a particular domain. Following this, the actual extraction of the questions are carried out:

A. From the textbook, by manually extracting questions from the exercise section of the textbook, that we consider the mini-world reference and hand-labelling them.
B. Manual extraction and labelling of questions from past question papers and quizzes published by PES.
C. By using the Stack Exchange API [3] to retrieve questions based on keywords extracted from the mini-world reference. These questions will be for validation and testing. In order to speed up this process, either Map-Reduce or threads can be considered. Through this process, collect a large number of questions (around 3 million).

II. Training

Train classifiers to accurately output a probability distribution of the various levels a question can belong to, based on the seeded data. Assume an independence between the cognitive domain and the cognitive processes dimensions. [3]

A. For the Knowledge domain, a combination of Naive Bayesian-based document classification and LSA/LDA/Doc2Vec/GloVe [4] based semantic similarity is performed to gauge how similar a question is to a particular section. The

question is first matched to the correct section with a degree of confidence, and then semantic analysis is carried out to see how much the question resembles the section. The hypothesis is that if the question has high affinity to the section, it is a relatively easier question.

B. For the cognitive dimension, an ensemble of the Support Vector Machine (SVM), Bidirectional Recurrent Neural Networks (BRNNs) and Multinomial Naïve Bayes Classifier (MNBC) has been built. A set of around 450 keywords have been collected and segregated into the respective Cognitive domains. A TFIDF based filtration technique is employed to filter out noise and retain only the keywords and relevant reference context that would help the classifier determine the correct skill level for that question.

III. Displaying Results

Display the predicted difficulty for unknown questions not seen in the dataset. This is shown in the form of a heat map.

IV. Use Case Evaluation

The system described in Steps I through III should be used to evaluate at least one of the three use cases described above.

1.3. Acknowledgement

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

PES INSTITUTE OF TECHNOLOGY

(An Autonomous Institute under VTU, Belgaum)

100 Feet Ring Road, BSK- III Stage, Bangalore – 560 085

Project ID: PW - 023

Project Title: Assessment of Question Quality using Bloom's Taxonomy

Project Team:

Mohit Surana	**1PI13CS092**
Shiva Karnad Deviah	**1PI13CS147**
Shrey Agarwal	**1PI13CS150**

This project report was submitted for review on _____. I acknowledge that the project team has implemented all recommended changes in the project report.

Guide signature with date:

Guide Name: Prof. Nitin V Pujari

CHAPTER 2

LITERATURE SURVEY

2. Literature Survey

As part of literature survey, we first looked at Bloom's Taxonomy, what the motivation for it was, various improvements made to the original model and its applications. Then we looked at past work done to replace manual intervention in the task of classification of these questions and the reasons to use the same.

Bloom's Taxonomy came into existence in 1956 and slowly it began to be used as a pedagogical tool to assess the various levels of knowledge or skill that was required to answer questions. This helped evaluators to design papers to suit the level of their target audience and also helped them to assess the weak areas in need of attention. This was used in conjunction with specialized exercises that were shown to help improve the different cognitive levels of knowledge of a given person with the aim to eventually lead to their better performance.

In this Taxonomy, the Knowledge category included both noun and verb aspects. This brought one-dimensionality to the framework at the cost of a Knowledge category that was dual in nature. This anomaly was eliminated in the Revised Taxonomy. Introduced in 2001 by Anderson, Krathwohl, et al, the Revised Taxonomy allowed the two aspects, the noun and verb, to form separate dimensions — the noun providing the basis for the Knowledge dimension and the verb forming the basis for the Cognitive Process dimension.

Although the possible benefits of using this are plentiful, its application is not widespread due to the requirement of intensive and skilled manual labour in order to classify the content. Many of the studies that were conducted required the support of domain experts to label the data and even amongst them there was often no unanimity. This has prevented scalable adoption of the Bloom's Taxonomy and ultimately the loss of the opportunity to reap its benefits.

In the recent past,

- Van Hoeij et al (2004) applied Bloom's taxonomy on essay-based questions but they recorded very low accuracy.
- Chang and Chung (2009) achieved high accuracy but they used an extremely small dataset.
- Yusof and Hui (2010) implemented an Artificial Neural Network (ANN) approach using multiple feature methods but were unable to get good accuracy.
- Haris and Omar (2012) used a rule-based classifier which was time consuming, expensive to make and maintain, limited to computer subjects, and performed badly in other domains.

Apart from the above papers, there were a few that showed promising results. Two of these research publications have originated from our very own CS department at PES University. Our guide and Head of Department, Prof. Nitin V Pujari, has done a lot of research and has made a lot of headway in this field.

- ***A Tutor Assisting Novel Electronic Framework for Qualitative Analysis of a Question Bank*** *(Nitin V Pujari, Nagashree Iyer)* [5]
    - This paper employed a new method of document classification that could classify questions to sections of the textbook with up to 90% accuracy. In addition to this, it would output a confidence score, and this score had a direct correlation to the difficulty level of the question. In this project, this work has been extended to include Bloom's Taxonomy.

- ***Application of Bloom's Taxonomy in day to day Examinations*** *(Nitin V Pujari, Bhargav HS, Gangadhar Akalwadi)* [6]
    - This paper applied the taxonomic principles to analysis of question papers, along the Cognitive level. A brief proof of concept was also created, whereby keyword based classification was employed to classify the questions. The authors collected around 200 keywords such as "what", "define", "explain" and "rationalise" – words that would give an indication as to what kind of question it was, and would then attempt to classify based on this. This method

achieved a good level of accuracy as well, and we have also improved on this technique.

- ***Exam Questions Classification Based on Bloom's Taxonomy Cognitive Level using Classifiers Combination*** *(Abdulhadi, Jabbar, and Omar)* [7]
  - Explored different concepts such as SVM, NBC, and K-nearest neighbours with good results: around 80% accuracy

- ***Automatic Classification of Questions into Bloom's Cognitive Levels using Support Vector Machines*** *(Yahya and Osman)* [8]
  - Used SVM with good results: around 80% accuracy

- ***Automatic Classification of Answers to Discussion Forums According to the Cognitive Domain of Bloom's Taxonomy using Text Mining and a Bayesian Classifier*** *(Pincay, Ochoa)* [9]
  - Used NBC, but achieved a moderate degree of precision

- ***Classifications of Exam Questions using Linguistically-Motivated Features: A Case Study Based on Bloom's Taxonomy*** *(Addin Osman, Anwar Ali Yahya)* [10]
  - Used SVM, NBC, Logistic Regression and Decision Trees and used a combination voting strategy to pick the class from the predictions made by the classifiers.

- ***Programming Exam Questions Classification Based On Bloom's Taxonomy Using Grammatical Rules*** (Dhuha A. Abduljabbar, Ghadah Al-Khafaji and Nazlia Omar) [11]
  - Used a combination of keyword based approach combined with a set of POS based rules to achieve an average F1 score of 87%

- ***Bloom's Taxonomy Question Categorization Using Rules And N-Gram Approach*** (Syahidah Sufi Haris and Nazlia Omar) [12]
  - Parsed the questions and matched it against a set of 64 rules to determine the

cognitive levels for a set of 135 questions

All work that we found involved assessment on the cognitive domain and was, thus, restricted to one-dimensional analysis only. This prevents us from having a finer grained assessment. We wish to tackle this by evaluating on a two-dimensional grid formed by cognitive and the knowledge domain. Most of the past work had very low values of recall and we wish to improve upon that by making use of better algorithms and representations.

CHAPTER 3

SYSTEM REQUIREMENTS SPECIFICATION

3. System Requirements Specification

3.1. High level block diagram of the solution

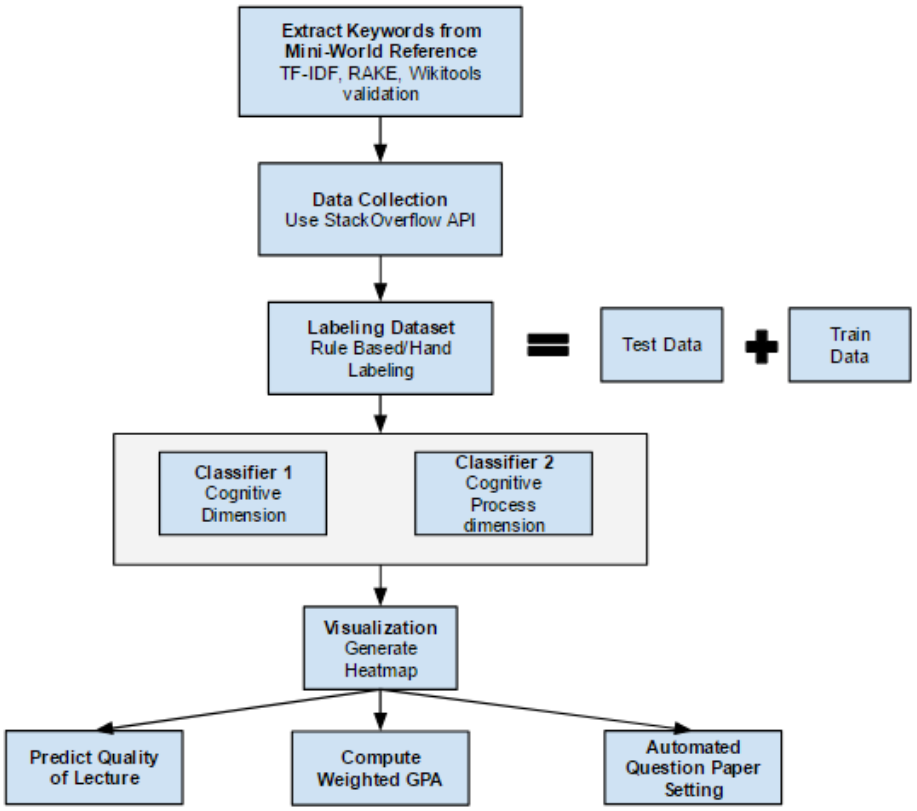

Figure 3-1- High Level Solution

3.2. Environment Used in the Project

The project was developed on Windows and Mac OS and hence is compatible for the same. The whole code is developed in Python 3.6 and requires packages like scikit-learn, that have been listed in section 3.2.2.

3.2.1. Hardware Required

In order to exploit the full power of the Map-Reduce framework for dataset collection, a server cluster is required. This an optional requirement. In the absence of a cluster, i.e., in standalone mode, a threaded approach to dataset collection will be faster due to less overhead.

3.2.2. Software Required

Listed below is all of the software tools required to complete this project:

1. python3
2. Xpdf and mupdf for PDF parsing
3. Various 3rd party python3 packages
    - sklearn (scikit-learn) library for ML classifiers
    - gensim and nltk for NLP tasks
    - numpy and scipy for mathematical operations
    - matplotlib, pandas, and seaborn for graphs
    - tkinter for the minimalistic interactive GUI
    - Django framework for dashboard UI
4. Latest web browser to view the dashboard
5. Dashboard dependencies (provided by the Django server)
    - Bootstrap and JQuery
    - Highcharts JS library
    - Chart.Heatmap library

3.2.3. Requirements for the Project

3.2.3.1. Functional Requirements

F1	Keyword Extraction	Given a mini-world reference, the system should adequately extract the most meaningful keywords in that context.
F2	Stack Overflow Question Extraction	The system should leverage the APIs exposed by Stack Overflow and partner websites to fetch a number of questions associated with the provided keywords.
F3	Question Paper Extraction	The system should also be able extract questions from a given question paper in a particular format.
F4	Assessing Question Quality	The system should be able to classify questions according to Bloom's Modified Taxonomy, displaying a heat map.
F5	Question Paper Analysis	The system should analyse a given question paper and provide a heat map indicating the overall level of the paper.

Table 3-1- Functional Requirements

3.2.3.2. Project UI

U1	Interactive User Experience	A Graphical User interface should be provided where a user should be able to enter a question (or a list of questions).
U2	Heat Map	When the *Submit* button is clicked, a new window should appear which shows a heat map on the Bloom's Matrix for the question.
U3	Teacher Dashboard	A Graphical User Interface for the teacher where he/she can track the performance of students for a particular question paper and the analysis of question paper set by him/her.

Table 3-2- Project UI

3.2.3.3. Non-Functional Requirements

NF1	Aesthetic frontend (usability)	The front end for the application must not suffocate the users with too much information, and should be intuitive and user-friendly.
NF2	Documentation	Should be provided with a simple yet comprehensive usage manual. Code should have sufficient comments so that future development and maintenance is easier.
NF3	High backend performance	Should implement the most efficient and appropriate algorithms for each and every sub task.
NF4	Maintainability	All features should be designed in a modular form to allow easy maintenance.
NF5	Response time	Response time should not be too long on an average.
NF6	Reusability	The classifier when developed for a particular subject, should be migratable to suit the requirements of other similar subjects without much hassle.

Table 3-3- Non-Functional Requirements

3.2.4. Constraints and Dependencies

The constraints that we faced during the planning and implementation of this project consisted of the following:

- The StackExchange API has a daily limit of 300 requests per day and a rate limit of 30 requests/second.
- The nature of our project is such that the training and validation processes are domain dependent. Two subjects have been considered as a proof of concept - *Algorithms* and *Operating Systems*.
- Since the topic we have considered for our project has never been ventured before, the dataset was not readily available and we had to manually label data to create training set. As mentioned before, questions have been extracted through various means, the most significant being the manual extraction and labelling of questions from the textbook. The hand-labelling process involves having two "experts" independently label the questions as per Bloom's

Taxonomy along both the Knowledge and Cognitive dimensions.

- Validating the results of our project was also a major challenge and the only solution was to perform human recall.

Dependencies include the following:

1. Upstream dependencies:
    - StackExchange and partner websites
    - All the software mentioned in Section 3.2.2.
2. Downstream dependencies:
    - End users such as teachers, students and educational institutions

3.2.5. Assumptions

- For training, we will assume independence between the two dimensions on Bloom's Matrix. This makes for easier classification and identification of hierarchical structure amongst classes which can be captured easier if they are treated independently.
- We also assume that the software mentioned in Section 3.2.2 being used is bug free.

3.2.6. Use Case Diagram

Below is the use case diagram for our system, indicating some of the many possible use cases our system can be applied to. We have only implemented the assessment and heat map generation use cases.

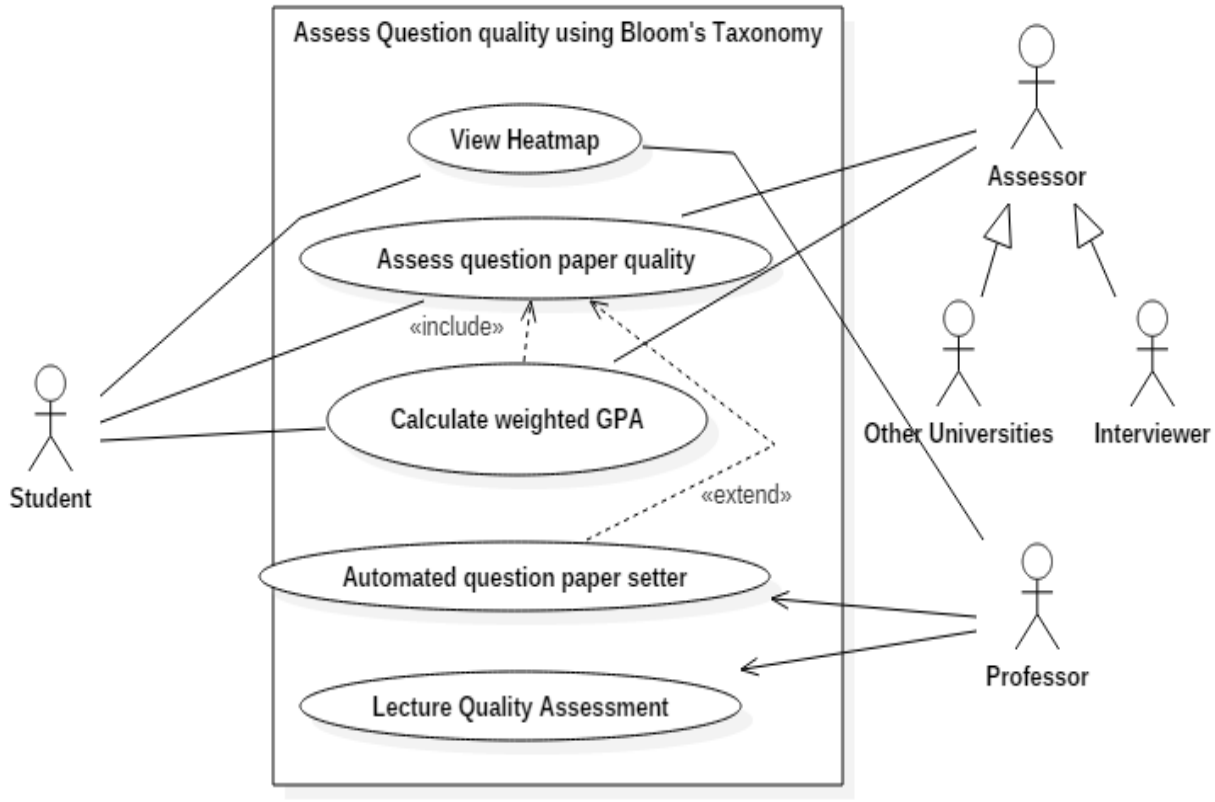

Figure 3-2- Use Case Diagram for the question quality assessment system

3.2.7. Requirement Traceability Matrix

Req ID	Description	Assignee	Test Plan	Test IDs
F1	Keyword Extraction	Shiva, Shrey	Will verify the quality of keywords extracted using the glossary of the textbook.	T1
F2	Stack Overflow Question Extraction	Shiva	A list of keyword : question ID pairs will be maintained that can be used to check if there are any issues.	T2
F3	Question Paper Extraction	Shrey	The question papers provided by the institute will be parsed by us and with the help of exception handling, we will detect parse errors.	T3
F4	Assessing Question Quality	Mohit, Shiva, Shrey	A labelled dataset (BCLs) that has been created by the authors of one of the papers cited in the Literature Survey will be used for the initial training. Further training will be done by using data labelled by us after discussion with subject teachers.	T4
F6	Question Paper Analysis	Mohit, Shrey	The question paper setter will be asked to provide some indicative labels and other teachers for the same subject shall discuss and once consensus has been reached, we shall use that as the standard.	T6
U1	Interactive User Experience	Shiva, Mohit	Shall be tested by asking various users with differing levels of comfort in the usage of computers and their feedback will be used to improve or modify the provided user interface.	T7
U2	Heat Map	Mohit, Shrey	A comprehensive set of self-generated cases shall be used to see	T8

			if the right regions are being highlighted.	
U3	Teacher Dashboard	Mohit	A Graphical User Interface for the teacher where he/she can track the performance of students for a question paper and the analysis of question paper set by him/her.	T9

Table 3-4- Requirement Traceability Matrix

CHAPTER 4

SCHEDULE

4. Schedule

Below is a Gantt Chart representing the schedule of our project.

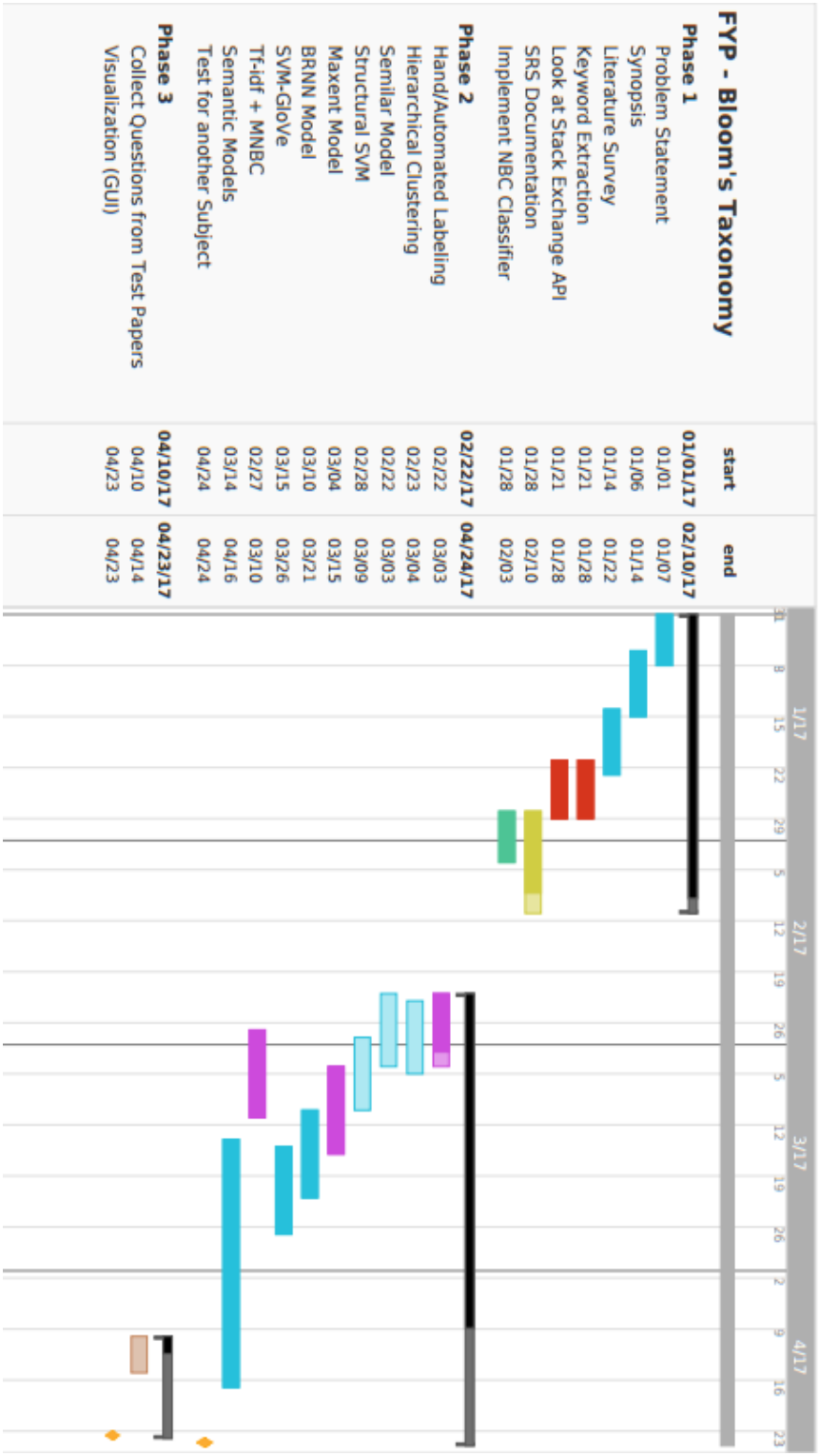

Figure 4-1- Gantt Chart

CHAPTER 5

SYSTEM DESIGN

5. System Design

5.1. High Level Design

A high-level design of the system is as follows:

5.1.1. Dataset Collection

The input data to our system is a question bank. For our system to be able to accurately classify questions, an adequate amount of training data is required. is collected from three main sources: the textbook, past question papers, and the Internet.

A) To collect questions from the textbook, it is required to convert the PDF into a form from which extracting the questions becomes easy. Once the textbook has been parsed, the questions can be manually extracted or automated.

B) Past question papers need to be procured, and the questions manually extracted and put into a csv or excel file.

C) Getting questions from Stack Overflow is a little more complicated, because we'd need a method to extract only those questions which are relevant to our subject. To do this, we need to do some form of questions scraping based on keywords that are relevant to us. Procedure to do this is as follows:

1. Keywords are extracted from the textbook using TFIDF and RAKE algorithms.
2. These keywords are validated using DBPedia to ensure that they are valid keywords (valid in the sense that they are completely relevant to the subject and not generic keywords).
3. We then make use of Stack Overflow API to get questions from Stack Overflow website which contain these keywords. The set of questions obtained now becomes our dataset.

5.1.2. Model Training and Creation

For the purpose of simplicity, the two dimensions are assumed to be independent. With this, we can now consider each one separately, using a different classification technique for both.

I) The Knowledge classifier

Prepare a model, such that, given a question, the model should be able to classify it into one of the sections of the textbook. This is important because we need to ensure that our system predicts the correct similarity score for the correct section, not for an incorrect one. This can be done with up to 95% accuracy.

Meanwhile, train a semantic similarity model on the mini world corpus, that is the textbook. With the knowledge of which section a question belongs to, we use one of these unsupervised models to retrieve the similarity score between the question and the section.

II) The Cognitive classifier

Train a model that should be able to identify what skill level is required to solve that question. Usually this is indicated by certain words such as "what", "define", "design", and "rationalise". Some sort of filtering strategy should be implemented to eliminate the fluff from a question, retaining only the information most important to identify the type of question. Some advanced pre-processing is needed for this. Following this step, we can then train the classifiers on hand labelled filtered questions.

5.1.3. User GUI

Provide an interactive UI to a user. Allow users to enter a question. Some amount of validation should be done to ensure that the question being entered belongs to the mini world, else the input should be rejected.

Once the input has been validated, all the required models should be loaded and ready to perform classification on the input. The results should be sent back to the user and displayed in the form of a probability heat map on a 2D matrix, in such a way that the user should be able to immediately interpret it.

5.2. Architectural Diagram

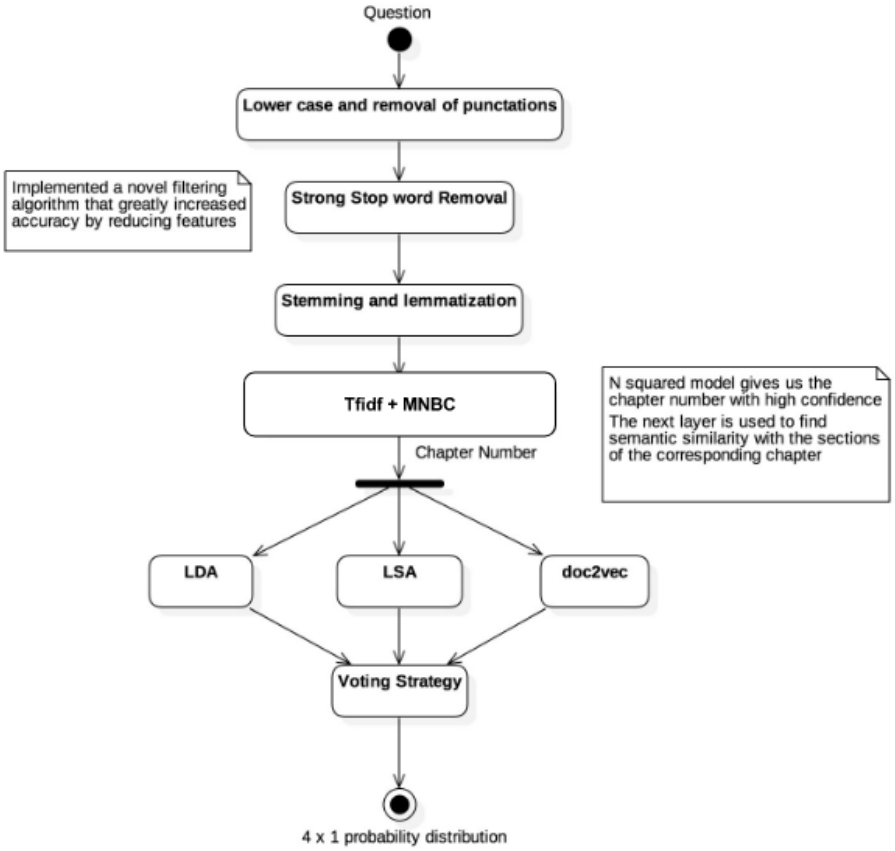

Figure 5-1- Classifier 1: Knowledge Dimension

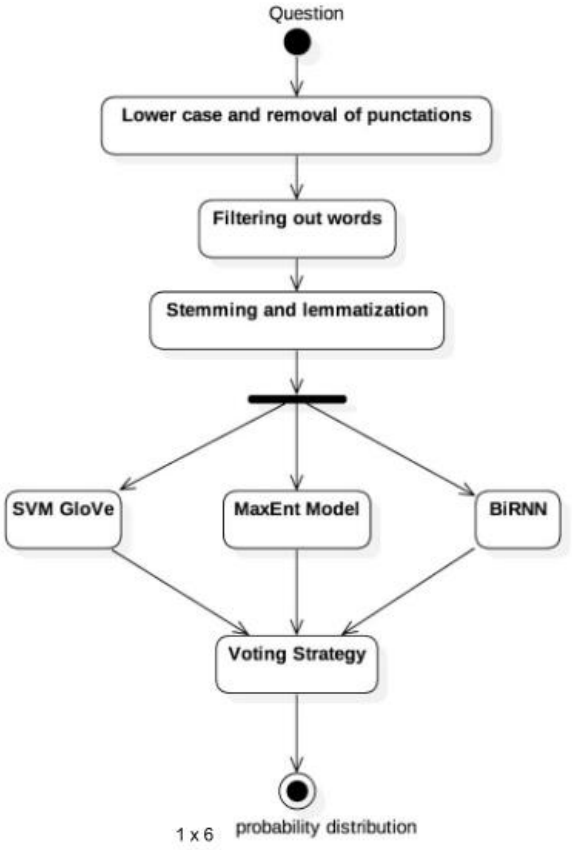

Figure 5-2- Classifier 2: Cognitive Dimension

5.3. Sequence diagram: Question paper analysis and Teacher Dashboard

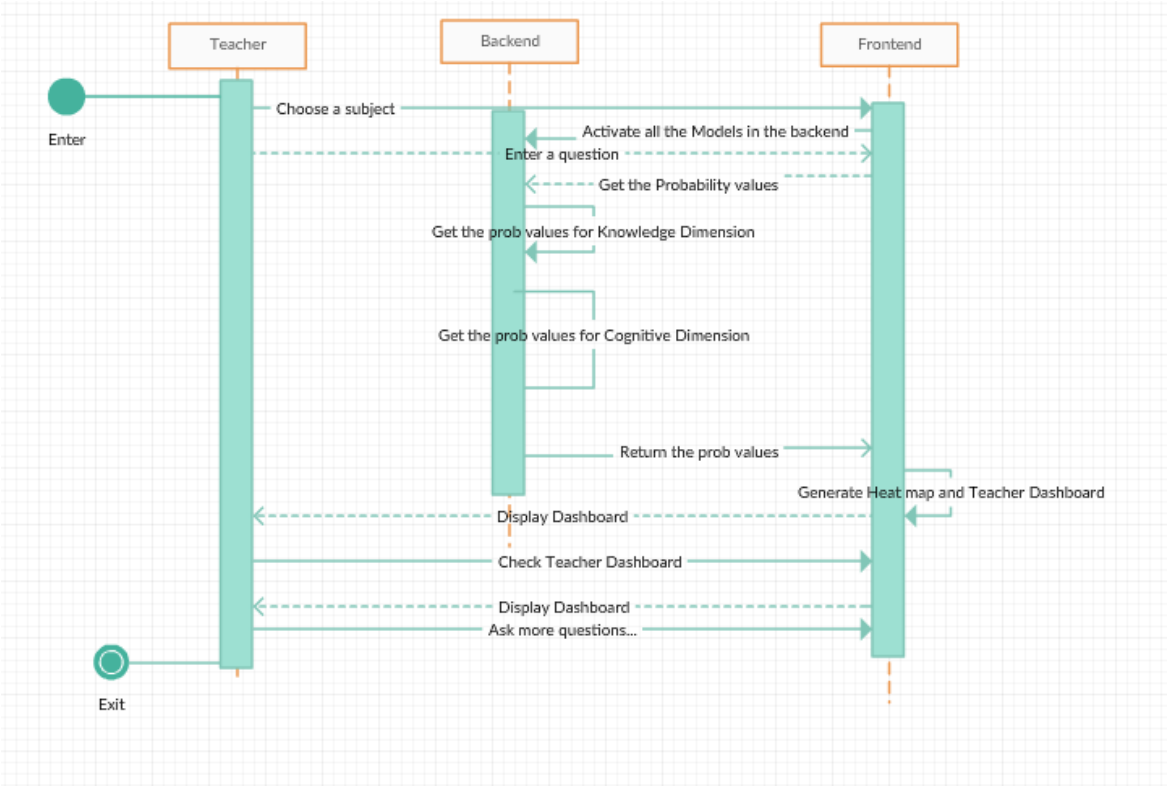

Figure 5-3- Sequence Diagram for Question Paper Analysis and Teacher Dashboard

5.4. UI Design

For the purpose of easily and quickly visualizing the outputs generated by the system, a GUI has been built. The GUI allows users to input a question into a text field. Once the user confirms, the question is then validated to check that it is a relevant question, i.e., it belongs to one of the two subjects that the system is currently planning to support as a proof of concept for this problem. This is done by examining the keywords (again, extracted using a pre-trained TFIDF model or RAKE technique), picking the most important ones and validating them using DBPedia. This ensures we have a valid input and so our outputs will be valid and meaningful as well.

The question is passed as input to each of the two models (one for knowledge and the

other for skill) and the outputs from each is obtained and combined by taking the dot product of the probability distributions. This result is a probability heat map which gives us an indication of what are the most probable levels the question belongs to on the taxonomic grid.

The heat map is displayed on the right-hand side of the screen, and on the left-hand of the screen is a list of questions that were previously asked. The user can click on any of these questions in the list and the heat map corresponding to that question (which was previously saved on generation) will again display on the right-hand side, with a small output field underneath the map indicating what question the heat map is currently being displayed for.

Another facet of the GUI is employing the system as a pedagogic tool to perform detailed analysis of students' performance in some examination or quiz. There will be a second screen allowing users to enter a csv file containing a list of questions. For each question, indicate for a group of students, how each student performed on that question (number of marks scored / right-wrong scheme). This csv is then parsed, each question processed for each student and numbers are crunched. The GUI then displays cumulative statistics indicating the distribution of questions according to the Knowledge and Skill dimensions, and the percentage of students of got those kinds of questions correct.

5.5. Updated RTM

Req ID	Requirement	Architectural (block / subsystem)	Design	Implementation	Testing
F1	Keyword Extraction	5.1.1			
F2	Stack Overflow Question Extraction	5.1.1			
F3	Question Paper Extraction	5.1.1			
F4	Assessing Question Quality	5.1.2			
F5	Question Paper Analysis	5.1.3			
U1	Interactive User Experience	5.1.3			
U2	Heat Map	5.1.3			
U3	Teacher Dashboard	5.1.3			

Table 5-1- Updated RTM after System Design

CHAPTER 6

DESIGN

6. Design

6.1. Data Flow Diagram

6.1.1. Data Flow Diagram – Level 0

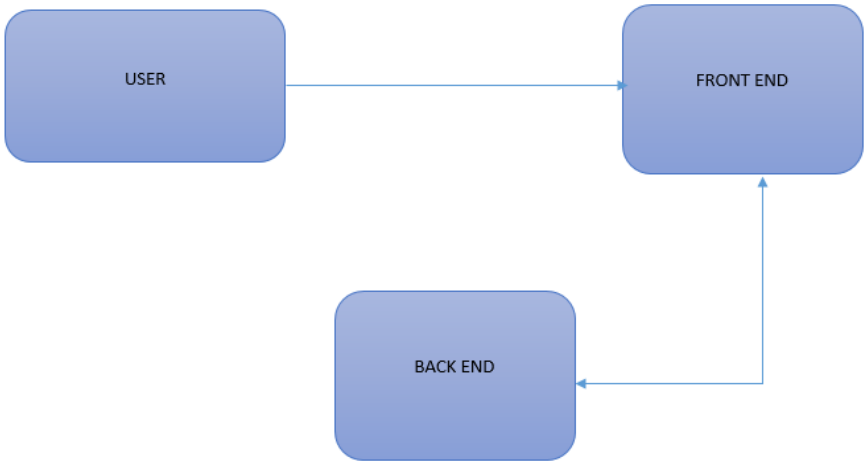

Figure 6-1- Data Flow Diagram – Level 0

6.1.2. Module Name: Back End

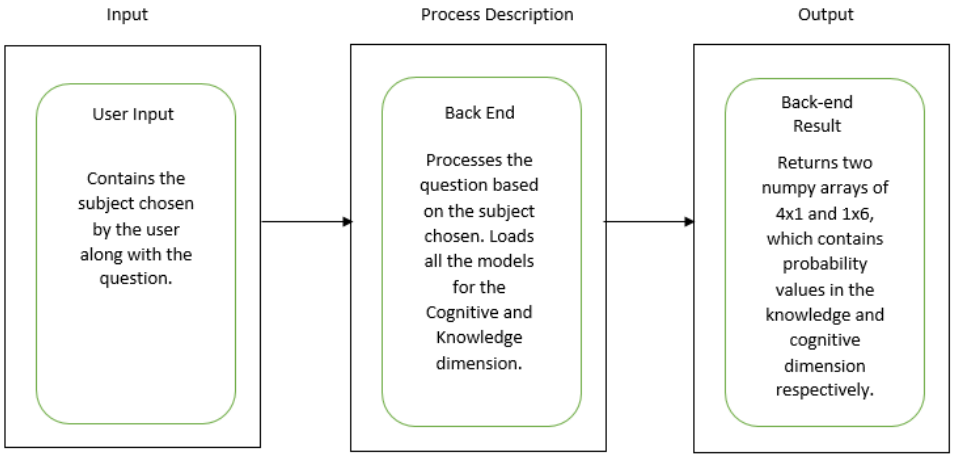

Figure 6-2- Backend Module

6.1.3. Module Name: Front End

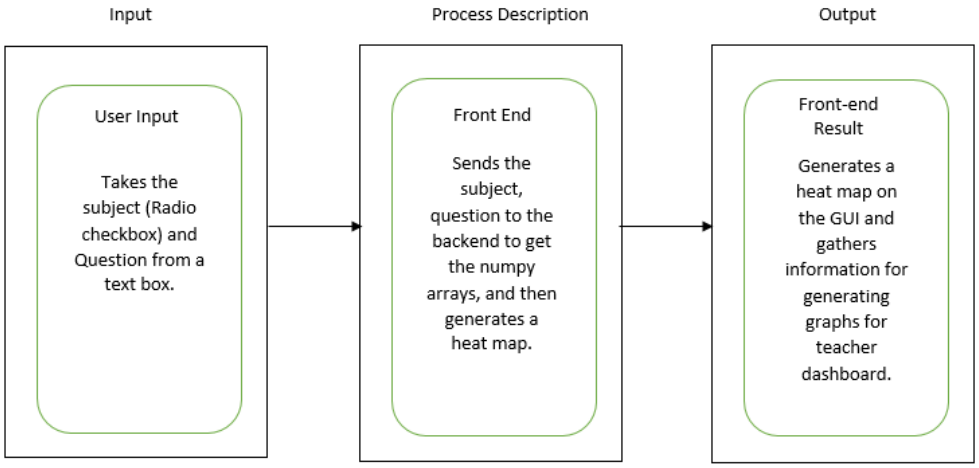

Figure 6-3- Front End Module

6.1.4. Data Flow Diagram – Level 1

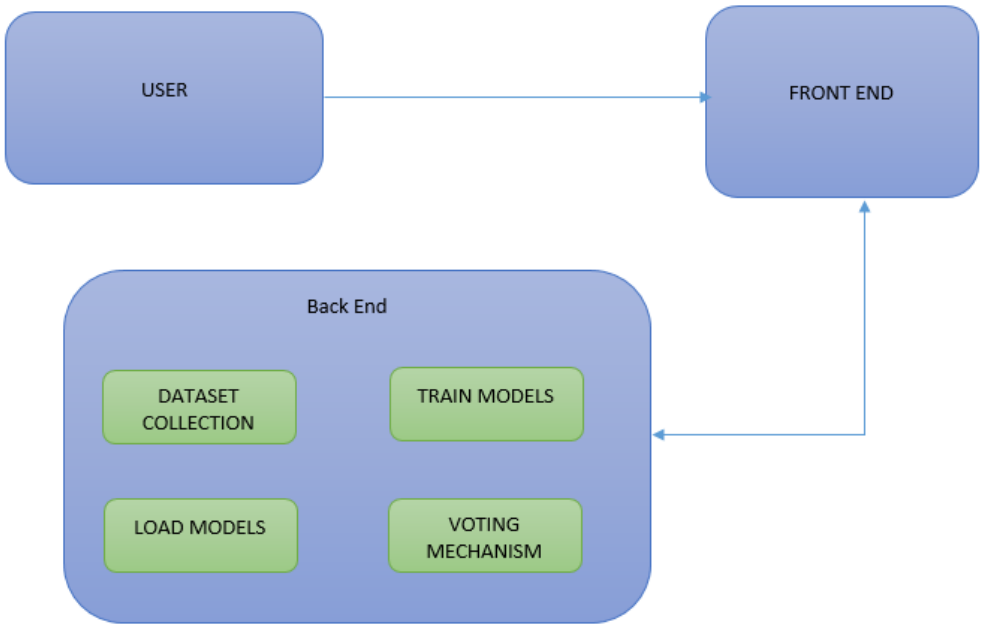

Figure 6-4- Data Flow Diagram - Level 1

6.1.5. Module Name: Dataset Collection

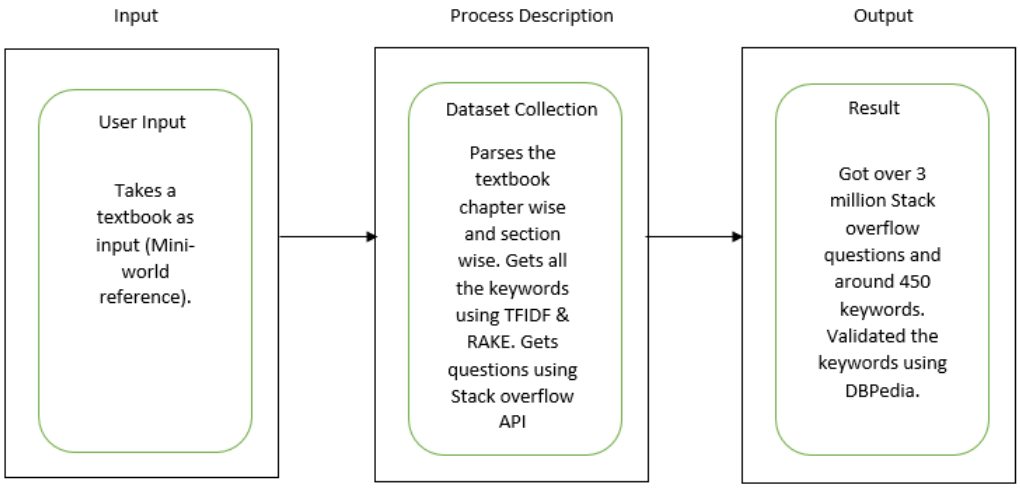

Figure 6-5- Dataset Collection Module

6.1.6. Module Name: Train Models

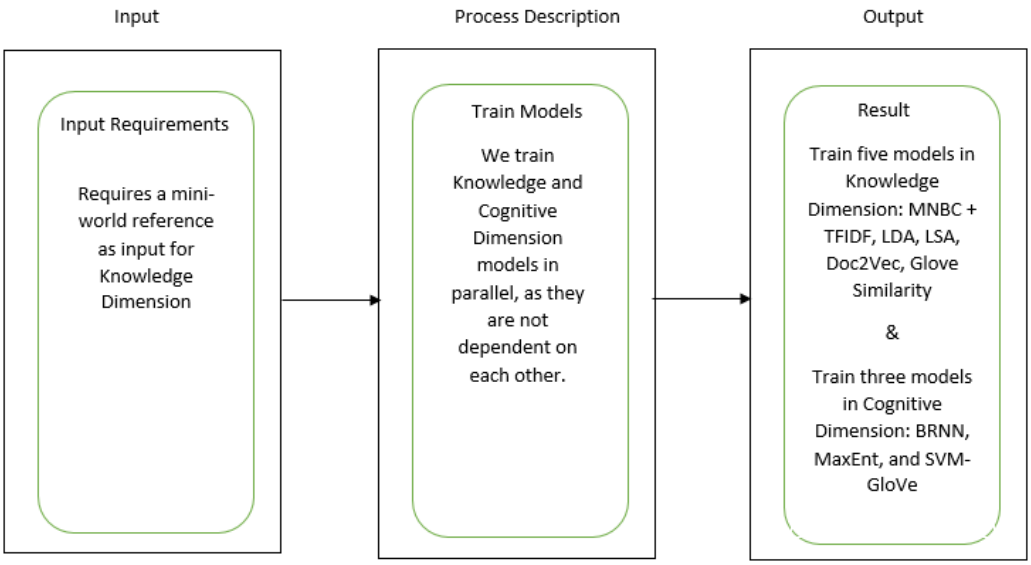

Figure 6-6- Train Models Module

6.1.7. Module Name: Load Models

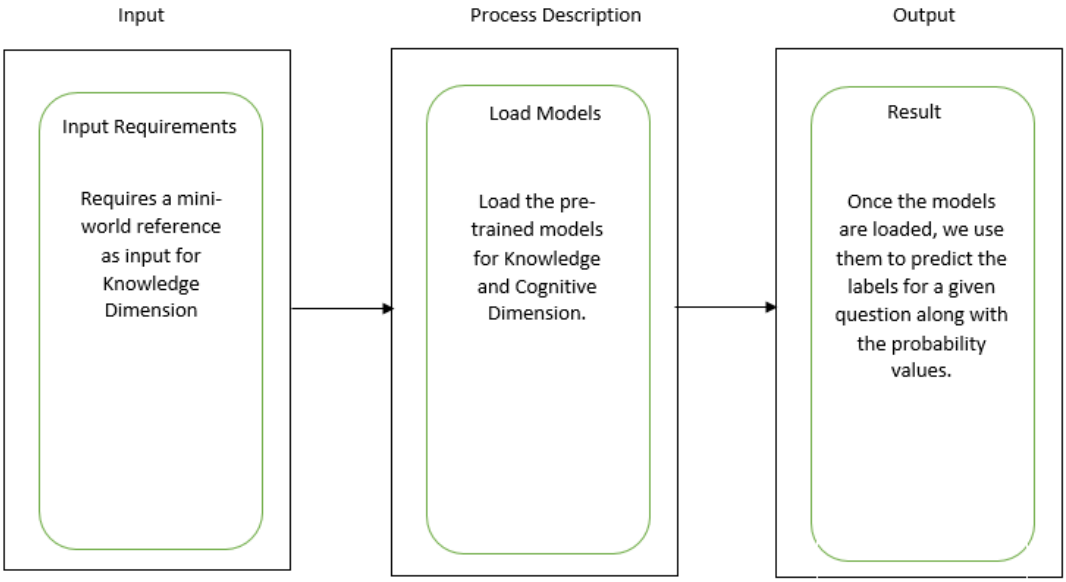

Figure 6-7- Load Models Module

6.1.8. Module Name: Voting Mechanism

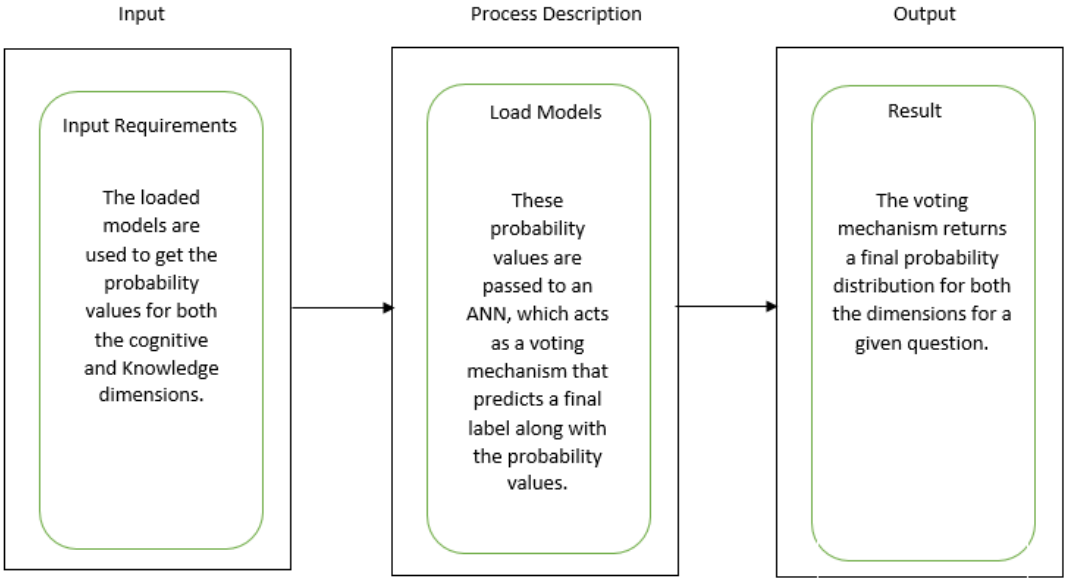

Figure 6-8- Voting Mechanism Module

6.1.9. Data Flow Diagram – Level 2

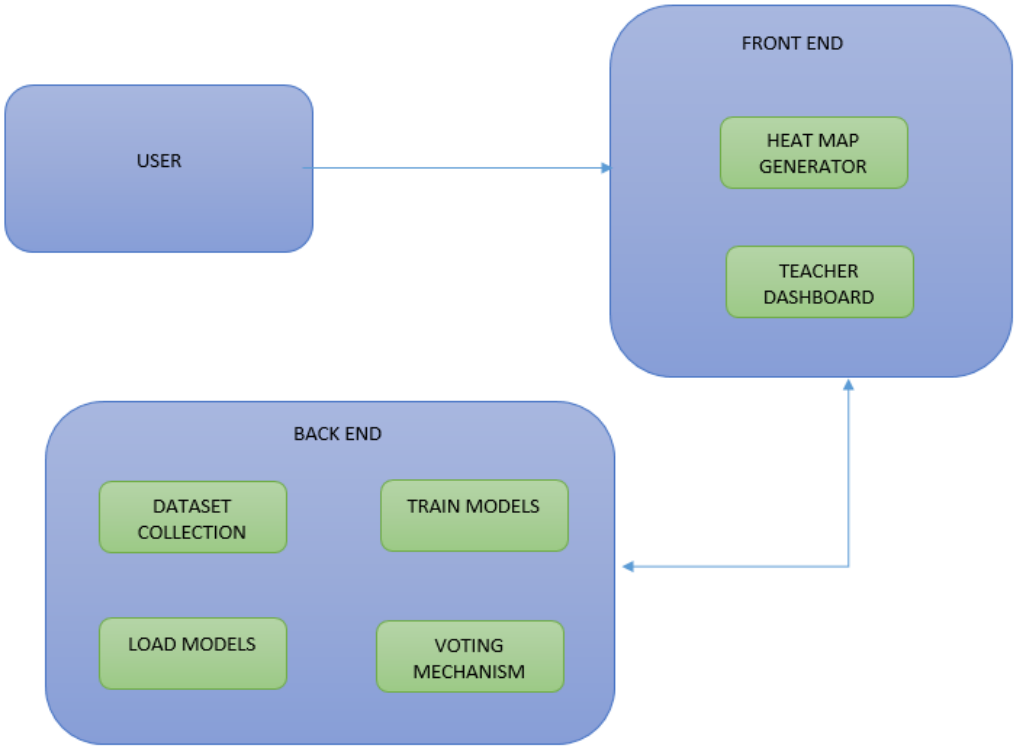

Figure 6-9- Data Flow Diagram – Level 2

6.1.10. Module Name: Heat Map Generator

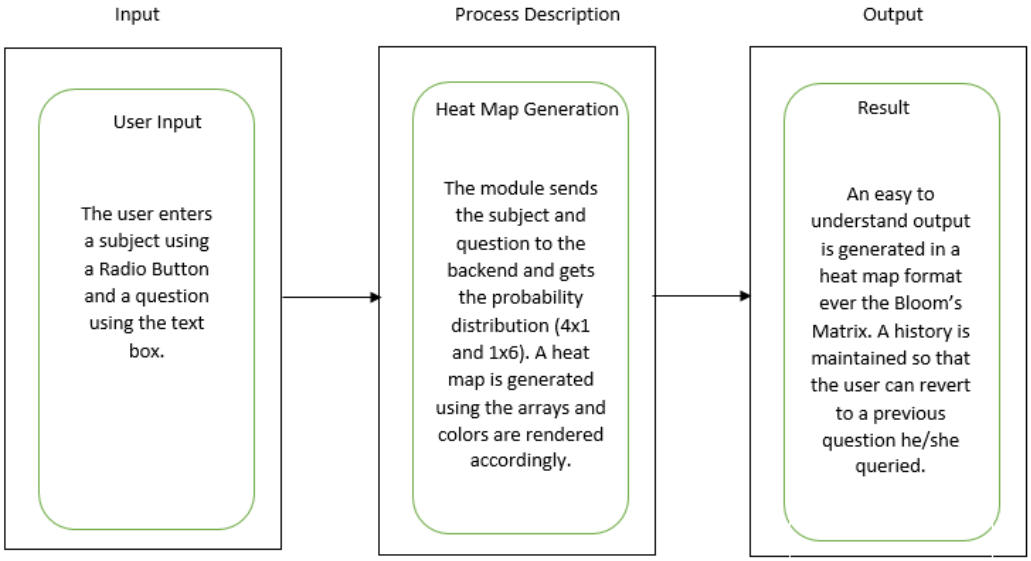

Figure 6-10- Heat Map Generator Module

6.1.11. Module Name: Teacher's Dashboard

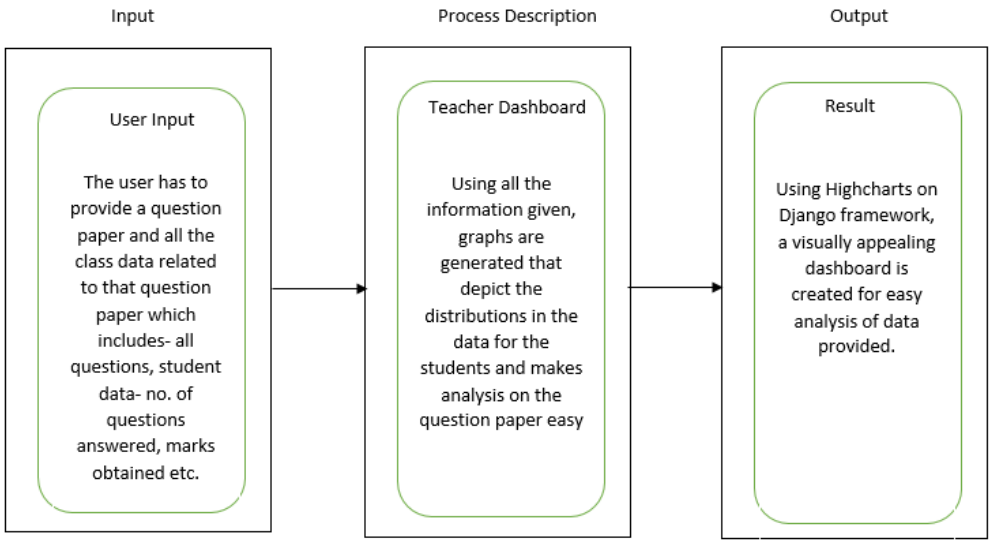

Figure 6-11- Teacher's Dashboard Module

6.2. Updated RTM

Req ID	Requirement	Architectural (block / subsystem)	Design	Implementation	Testing
F1	Keyword Extraction	5.1.1	6.1.5		
F2	Stack Overflow Question Extraction	5.1.1	6.1.5		
F3	Question Paper Extraction	5.1.1	6.1.5		
F4	Assessing Question Quality	5.1.2	6.1.2		
F5	Question Paper Analysis	5.1.3	6.1.2		
U1	Interactive User Experience	5.1.3	6.1.9		
U2	Heat Map	5.1.3	6.1.10		
U3	Teacher Dashboard	5.1.3	6.1.11		

Table 6-1- Updated RTM after Architectural Design

CHAPTER 7

IMPLEMENTATION

7. Implementation

7.1. Algorithms

In this section, we highlight the high-level algorithms used at each step of our project. Wherever necessary, we also detail the high-level algorithm of the machine learning / pre-processing method we employed for that process.

7.1.1. Dataset Generation

This section involves two parts: One is the generation of the dataset that will be used to train the classifiers. The second is the generation of the Stack Overflow dataset, that will be used to validate the credibility of the system.

7.1.1.1. Train Dataset Generation

1. Using a combination of Xpdf and mupdf, parse the PDF and generate separate text files for each section.
2. Walk through each section collecting any exercise questions present at the end.
3. Once questions have been collected, have 2 people hand label these questions. Each person is designated an expert and it is assumed the expert is knowledgeable in the field the questions belong to. Both experts must, independent of each other, label each question along the Knowledge and Cognitive domain separately.
4. If there are no discrepancies between the 2 sets of labels, skip this step. Otherwise, call in a 3rd expert to resolve the discrepancies.
5. Split the data into two sets of files, 70% of the original data will be used to train the models, while 30% will be used to carry out testing.

7.1.1.2. Stack Overflow Question Bank Generation

1. Train a TFIDF model on the textbook. The model should now have learned the most important words in the textbook.
    1. Use the pre-trained TFIDF model to pick the most relevant keywords with a

TFIDF value above a pre-set threshold. For the purpose of simplicity, we set the threshold to 0.6. All words with a value lower than this are ignored.

2. Use the RAKE technique to extract keywords from the textbook. We can specify that we want unigrams and bigrams from the textbook. RAKE will quickly determine the most important ones and return them.

2. For each extracted keyword, initialize a wikitools object in python and use it to establish a connection to DBPedia. This object can now query DBPedia to disambiguate a word. Based on this disambiguation, the keyword is either accepted or rejected.
3. For each validated keyword, call the Stack Overflow's Questions API with python's urllib module. The API returns up to 100 questions per request.
4. Once all questions for the keywords have been exhaustively collected, apply filtration technique to remove irrelevant questions (such as those pertaining to debugging some particular software – these appear a lot). This can again be done with a pre-trained TFIDF model or a model that learns and understands semantics, such as the LSA or LDA.

7.1.2. Training the Knowledge Classifier Model

1. Train the document classifier to predict the right section that a question belongs to.
2. Train a semantic similarity model (one of the 3 mentioned previously – LDA, LSA, or Doc2Vec) on the mini-world reference.
3. For each question, get the correct section with its confidence value. Also use the pretrained similarity model to convert the section and the question to its equivalent n-dimensional vector representation.
4. The concatenated vectors and the confidence value will become the input to a MLP (ANN), and the MLP trains on this input
5. Predict and print accuracy report.

7.1.2.1. Training the Document Classifier

1. Take a mini-world reference as input (a textbook essentially).
2. Train a TFIDF model using the mini-world reference.

3. Use the parsed chapters (done in dataset generation) to train MNBC classifier while making use of the TFIDF model. Save the models.
4. Train another MNBC model, this time use the parsed sections for each chapter. We do this to get a sense of hierarchy between the chapters and the sections.
5. We also train 4 semantic models to find a similarity score between sections and question. Namely: Latent Dirichlet Allocation (LDA), Latent Semantic Analysis (LSA), Doc2Vec, GloVe model.
    1. LDA: This model is trained on bag of words of sections.
    2. LSA: This model is trained on TFIDF values of the words.
    3. Doc2Vec: This is a simple model which works similar to word2Vec but works on a document level. (generates a vector for a given document)
    4. GloVe model: This model uses a pre-trained glove model which has been further customized on our mini-world reference.
6. Finally, a voting mechanism has been built in place to take the outputs from all the semantic models and predict a final probability matrix of 4x1.
7. So, when a question is passed to the knowledge classifier, the following happens:
    1. The question is first mapped to the right chapter using one of our pre-trained models.
    2. Once the best chapter is determined for the question, we use another MNBC model to find the best section match for the question.
    3. We pass this section along with the question to our various semantic models. Each model returns a semantic score which we use as our input for the Voting model (ANN) and get the final probability matrix of 4x1 along with the predicted label in the knowledge dimension.
8. Generate the accuracy of the model.

7.1.3. Training the Cognitive Classifier Model

1. Train the SVM-GloVe model, the MNBC model, and the BRNN model
2. Train an ensemble model which consists of the 3 models trained in Step 3 as constituents in the ensemble.

7.1.3.1. Training the SVM-Glove Model

1. Load the dataset in memory
2. For each question in the dataset, perform pre-processing and filtering based on pre-trained TFIDF models to remove words having TFIDF value beyond a certain threshold. This is by default, set to 0.25.
3. Load the pretrained GloVe model in memory
4. Train a TFIDF model on the textbook and keywords to learn word weights
5. For each question,
    a. For each word in the question, convert the word into its n-dimensional GloVe vector equivalent multiplied by its word weight.
    b. Take the mean of all the vectors in the question. This is the input
6. Train the SVM on this data
7. Predict and print accuracy report.

7.1.3.2. Training the MNBC Model

1. Load the dataset in memory
2. For each question in the dataset, perform pre-processing and filtering based on pre-trained TFIDF models to remove words having TFIDF value beyond a certain threshold. This is by default, set to 0.25.
3. Train a TFIDF model on the textbook and keywords to learn word weights
4. For each question, convert it into its corresponding TFIDF embedding.
5. Train the MNBC on this data
6. Predict and print accuracy report

7.1.3.3. Training the Bidirectional Recurrent Neural Network

1. Load the dataset in memory
2. For each question in the dataset, perform pre-processing and filtering based on pre-trained TFIDF models to remove words having TFIDF value beyond a certain threshold. This is by default, set to 0.25.

3. Load the pre-trained GloVe model in memory
4. For each question, and for each word in the question, convert the word into its n-dimensional GloVe vector equivalent multiplied by its word weight.
5. Train the BRNN. The training process is as follows:

   At every epoch, and for each input in the dataset, carry out forward pass on the vectors, get the error, and then backpropagate through time to make the BRNN learn its weights.
6. Predict and print accuracy report

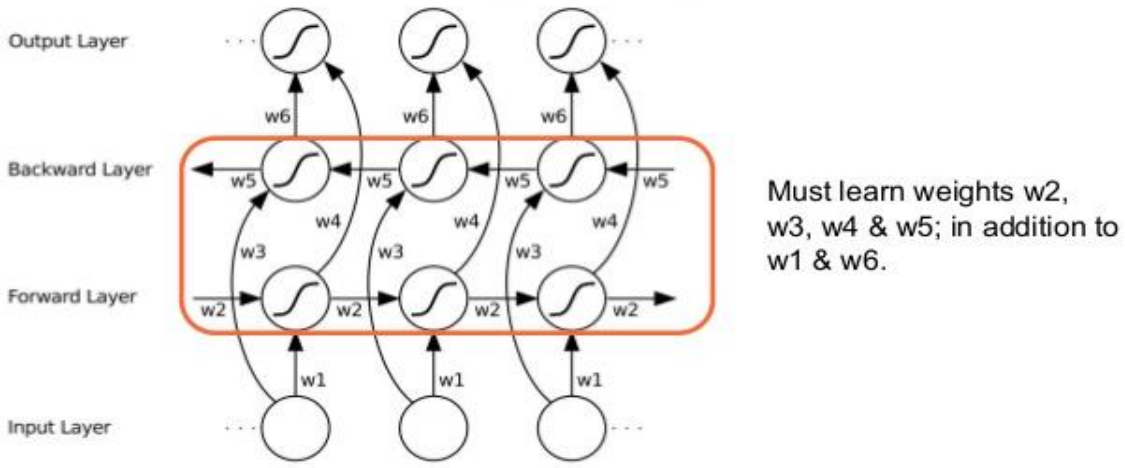

Fig 71: Flow diagram of BRNN

7.1.4. Visualization

1. In the beginning, the program loads all the models for cognitive and knowledge dimension.
2. The program takes a subject as command line parameter. *(*could be either *ADA*, *OS*, or any other custom subject you created by referring to the guide in Section 12.1.).
3. A window pops up (built using Tkinter) and provides a textbox.
4. When a question is entered in the textbox and submitted using the "Submit" button, the program calls the cognitive and knowledge classifiers.
5. The classifiers then return 4x1 and 1x6 numpy arrays.
6. We take the dot product of the arrays and with the help of gradient colors, we plot a

heatmap over the Bloom's Matrix.

7. The page can be refreshed and another question can be entered and the process repeats.

7.1.5. Dashboard

1. We make use of Django framework and with the help of HighCharts, render some graphs for analysing the paper quality and classroom statistics.
2. We also replicated the heatmap generated in the section 7.1.4 providing better features and much more easy and comfortable GUI.
3. A dashboard is created for the teacher where she can view various graphs that we plot using the information we get by analysing the question paper.
4. The program also requires the teacher/user to enter class data for the particular question paper which we use to plot some histograms.

7.2. Codebase structure

Below is shown the codebase structure for our GitHub repository. For the sake of compactness, only the directories have been listed. Files and models have not been shown.

```
src
├── GloVe_code
│   ├── build
│   └── src
├── Web-UX
│   ├── qqa
│   │   └── __pycache__
│   └── reports
│       ├── __pycache__
│       ├── migrations
│       │   └── __pycache__
│       ├── static
│       │   └── reports
│       └── templates
│           └── reports
├── __pycache__
├── datasets
├── models
│   ├── BiRNN
│   ├── MNBC
│   ├── Nsquared
│   │   ├── ADA
│   │   └── OS
│   └── SVM
├── preprocessing_code
│   └── __pycache__
└── resources
    ├── ADA
    ├── GloVe
    └── OS
```

Figure 7-2- Codebase structure

- The *src* folder is the root folder which contains all our programs, datasets, models and resources.
- The *GloVe_code* directory contains resources that allow us to train custom GloVe models with our textbook.
- The *Web-UX* directory contains everything that is needed to get the system up and running in a web browser with the power of Django.
- The *datasets* folder houses all the datasets used to train the classifiers – this includes the ADA question bank, OS question bank and various other datasets with which experiments were carried out.
- The *models* folder holds all the models that are worthy of being persisted.
- The *preprocessing_code* folder is a subdirectory to hold all the PDF parsing and extraction code. This is the only part of the system that must be manually reworked every time a new subject is to be introduced into the system. The pre-processing is not a seamless process as it involves catering to the idiosyncrasies of each and every textbook which is unique in some aspect or the other.
- The *resources* folder contains all the pre-trained GloVe models, PDFs, and section wise text files that are needed along with the datasets to train the models.

7.3. Coding Guidelines Used

To foster efficient development and features like readability and code reusability, we have set down some basic guidelines to be followed. They are as follows:

- Acknowledging the fact that strictly following this may be difficult, adherence to the guidelines is recommended but not mandatory.
- All developers in the team share ownership of the code and should strive to not only make it robust and efficient but also readable and self-documented.
- Comments should be added to explain functionality and can be omitted for the styling.
- To prevent problems due to indentation, we shall use 4 spaces instead of tabs.

- Convention for naming identifiers:

```
all_small_and_separated_by_underscores
```

- Convention for naming constants:

```
ALL_CAPITALS_AND_SEPARATED_BY_UNDERSCORES
```

- Imports should be sorted according to PEP8.
- Imports from different packages should be on separate lines.

```
import os
import sys
```

  Instead of

```
import os, sys
```

- Loop invariants should never be recalculated. In general, code should be optimized as part of initial coding instead of after refactoring.
- No important numbers should be hardcoded if they are to be used multiple times.
- Usage of an IDE like PyCharm Community Edition will help us follow the PEP8 rules and guidelines.

7.4. Sample Code

Below is a representative code snippet of one of the many classifiers coded during the project. This code piece has been chosen as it contains most of the features discussed in Section 7.3

```
import os
import random
import pickle

from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.base import BaseEstimator, ClassifierMixin
import numpy as np
from sklearn.pipeline import Pipeline
from sklearn.naive_bayes import MultinomialNB
from sklearn.externals import joblib
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split, cross_val_score, KFold
```

```
from utils import clean_no_stopwords, get_data_for_cognitive_classifiers

TRAIN = True

mapping_cog = {'Remember': 0, 'Understand': 1, 'Apply': 2, 'Analyse': 3, 'Evaluate': 4,
'Create': 5}

domain = pickle.load(open(os.path.join(os.path.dirname(__file__), 'resources/domain.pkl'),
'rb'))

keywords = set()
keyword_doc = ['' for i in range(len(mapping_cog))]
for k in domain:
    for word in domain[k]:
        cleaned_word = clean_no_stopwords(word, lemmatize=False, stem=False, as_list=False)
        keywords.add(cleaned_word)
        keyword_doc[mapping_cog[k]] += cleaned_word + '. '

class MNBC(BaseEstimator, ClassifierMixin):
    def __init__(self, tfidf_ngram_range=(1, 2), mnbc_alpha=.05):
        self.tfidf_ngram_range = tfidf_ngram_range
        self.mnbc_alpha = mnbc_alpha

        tfidf = TfidfVectorizer(norm='l2',
                                min_df=1,
                                decode_error="ignore",
                                use_idf=False,
                                sublinear_tf=True,)

        self.clf = Pipeline([ ('vectorizer', tfidf),
                              ('classifier', MultinomialNB(alpha=self.mnbc_alpha))])

    def __prep_data(self, X):
        if type(X[0]) == type([]):
            return [' '.join(x) for x in X]

        return X

    def fit(self, X, Y):
        docs_train = ['' for i in range(len(mapping_cog))]

        for x, y in zip(X, Y):
            docs_train[y] += ' '.join(x) + '. '

        for i in range(len(docs_train)):
            docs_train[i] += keyword_doc[i]
```

```
    tfidf = self.clf.named_steps['vectorizer']

    self.clf.fit(self.__prep_data(X), Y)

def predict(self, X):
    return self.clf.predict(self.__prep_data(X))

def predict_proba(self, X):
    return self.clf.predict_proba(self.__prep_data(X))
```

7.5. Unit Test Cases

We have manually tested all the functions. Here are few examples that we could document.

Test case ID	Test Case Description	Expected Result	Actual Result	Status (Pass / Fail)	Remarks
T1	predict_know_label(), knowledge dimension	returns 4x1 array and a label (integer)	Returned 4x1 array and a label (integer)	Pass	Validated the label, cannot validate the array.
T2	predict_cog_label(), cognitive dimension	returns 4x1 array and a label (integer)	Returned 4x1 array and a label (integer)	Pass	Validated the label, cannot validate the array.
T3	get_filtered_questions()	Should return first and last sentence	Returns first and last sentence	Pass	Works fine
T4	load_texts()	Should return texts and documents	Returns texts and documents	Pass	Works fine for any subject
T5	clean()	Should return a string with all stopwords removedand lemmatised	Got a clean, lemmatised version of question.	Pass	-
T6	get_glove_vectors()	Should return a dictionary with all the glove vectors	Got a dictionary. Same as expected	Pass	Works fine

T7	predict_proba()	Should return the average values for all the prob matrices	Returns an array with average values	Pass	-

Table 7-1- Unit Test Cases

7.6. Metrics for Unit Test Cases

1. We have over 100 functions in all our classes and python files. We have manually tested 90% of them. Some of them had to work for other functions to work (dependent functions) and hence testing for them was not required.
2. Given the nature of the project, we cannot afford any of our test cases to fail and hence we achieve over 90% test case successes.
3. Our code coverage is around 84%.

7.7. Updated RTM

Req ID	Requirement	Architectural (block / subsystem)	Design	Implementation	Testing
F1	Keyword Extraction	5.1.1	6.1.5	7.1.1.1	
F2	Stack Overflow Question Extraction	5.1.1	6.1.5	7.1.1.2	
F3	Question Paper Extraction	5.1.1	6.1.5	7.1.1.2	
F4	Assessing Question Quality	5.1.2	6.1.2	7.1.5	
F5	Question Paper Analysis	5.1.3	6.1.2	7.1.5	
U1	Interactive User Experience	5.1.3	6.1.9	7.1.4	
U2	Heat Map	5.1.3	6.1.10	7.1.4	
U3	Teacher Dashboard	5.1.3	6.1.11	7.1.5	

Table 7-2- Updated RTM after Implementation

CHAPTER 8

TESTING

8. Testing

8.1. System / Functional Test Specifications

Test case ID	Test Case Description	Expected Result	Actual Result	Remarks
T1-a	Keyword extraction – from the ADA and OS textbooks	A set of keywords must be extracted	The keywords have been extracted successfully	Lot of extra spurious keywords
T1-b	DBpedia validation – extracted keywords to be filtered	Keywords should get filtered based on whether they belong to the domain or not	Keywords are successfully filtered	Final list is greatly condensed in comparison to the non-validated keyword list
T2-a	Stack Overflow API test – to check whether we are able to make calls to the Stack Overflow API	Simple call is made and we get the same result as we get by searching on the website	Test successful for fetching questions tagged with 'algorithms'	-
T2-b	Proxy check – all proxies should be working in parallel to maximize the use of the API's functionality and minimize the total time for fetching questions	All proxies are working and fetching data	Manual examination of logging statements helped us to confirm this	-
T3	Question Paper Extraction – all questions extracted	The questions are extracted and saved as a csv file	Extraction was successful	The number, and content of the questions was verified
T4-a	Assessing Question Quality – using a sample input question	The classifier should give us the knowledge and the cognitive levels for the given question.	The classification is successful with some degree of accuracy	The accuracy values have been tabulated in the Results & Discussions section

T4-b	Unit test cases for all subsystems	Each subsystem should be able to cater to all the unit tests mentioned in the Implementation section.	100% test cases passed	-
T5	Question Paper Analysis	The accuracy of our classifier is benchmarked against a paper hand labelled by the paper setter	The classifier performs well on the given set of questions	-
T6	Interactive User Experience	The user should be able to pick either of the two subjects and test questions in an interactive mode.	The user can carry these actions and obtain the labels as outputs	-
T7	Heat Map	The numerical values generated should be graphically represented to make it easy to visualize	Logged outputs from the classifiers were checked against the UI representation and they were matching	-
T8	Teacher Dashboard	The teacher should have a dashboard analysing the question paper set and the students' performance on the tests	A beautiful UI with the breakup of question levels and graphs representing student performance have been prepared	The performance values on the graph match the ones from a generated excel sheet

Table 8-1- System / Functional Test Specifications

8.2. Test Environment Used

Since most of the data we worked with was created or assimilated by us, we did not have a reference standard to compare to. With this in mind, we decided to take test different subsystems via human recall and optimized each subsystem before moving onto the next one.

8.3. Test Procedure

We considered the use of unit tests for regression testing and researched about automation of functional testing but ultimately chose to skip the same due to the following reasons:

- Our architecture and logic was always changing as we were experimenting with a lot of different approaches. This would require a lot of additional time spent on rewriting our tests.
- For this reason, we decided to go ahead with a human recall based approach in which we tested the outputs produced by subsystems to gain confidence in our code.
- A component (function) once tested may or may not require re testing.
    - If the component is in the lower most level - like a utility function, then it did not require retesting unless the code was changed.
    - If any other component, that makes use of these lower level components, is modified, we would test them again.
- The machine learning classifiers were benchmarked against either other using metrics like accuracy, precision, recall and F1 score.

8.4. Example Test Result

Stage Name	Input to this stage	Output from this stage
Initial input	Traffic lights are crucial for transport safety. Assuming you have access to number of waiting cars at a given intersection, design an algorithm to ensure that a given car waits no longer than n seconds	Same question encoded as a string
Lowercase	Traffic lights are crucial for transport safety. Assuming you have access to number of waiting cars at a given intersection, design an algorithm to ensure that a given car waits no longer than n seconds	traffic lights are crucial for transport safety. assuming you have access to number of waiting cars at a given intersection, design an algorithm to ensure that a given car waits no longer than n seconds
Punctuation removal	traffic lights are crucial for transport safety. assuming you have access to number of waiting cars at a given intersection, design an algorithm to ensure that a given car waits no longer than n seconds	traffic lights are crucial for transport safety assuming you have access to number of waiting cars at a given intersection design an algorithm to ensure that a given car waits no longer than n seconds
Keyword filtering	traffic lights are crucial for transport safety assuming you have access to number of waiting cars at a given intersection design an algorithm to ensure that a given car waits no longer than n seconds	Knowledge: crucial assuming access number design algorithm ensure given waits no longer than n seconds Cognitive: design
Knowledge dimension classification	crucial assuming access number design algorithm ensure given waits no longer than n seconds	Multiple 4x1 probability distributions across the knowledge dimension
Cognitive dimension classification	design	Multiple 1x6 probability distributions across the cognitive dimension
Voting systems	Multiple 4x1 vectors and 1x6 vectors representing the probability distributions along the two dimensions	A combined 4x6 probability distribution that calcucates the results of the two dimensions separately and then merges them using a dot operation
Final answer	4x6 probability distribution with element at 2,5 as the maximum	Procedural Create

Table 8-2- Example Test Result

8.5. Test Metrics

Metrics Parameters	Values
Was Successful	True
Test run count	16
Test Failure count	1
Test ignore count	0
Test Runtime in milliseconds	N/A
Errors Fixed	**1**

Table 8-5 (a)- Test Metrics

The following error was fixed:

Errors	Reason	Fix
StackExchange API	The API provides only 100 calls/day.	Used proxy servers to make calls to the API. We could get 3 million questions after that in a matter of minutes.

Table 8-5 (b)- Test Failure Fix

8.6. Updated RTM

Req ID	Requirement	Architectural (block / subsystem)	Design	Implementation	Testing
F1	Keyword Extraction	5.1.1	6.1.5	7.1.1.1	T1-a
F2	Stack Overflow Question Extraction	5.1.1	6.1.5	7.1.1.2	T2
F3	Question Paper Extraction	5.1.1	6.1.5	7.1.1.2	T3
F4	Assessing Question Quality	5.1.2	6.1.2	7.1.5	T4-a
F5	Question Paper Analysis	5.1.3	6.1.2	7.1.5	T5
U1	Interactive User Experience	5.1.3	6.1.9	7.1.4	T6
U2	Heat Map	5.1.3	6.1.10	7.1.4	T7
U3	Teacher Dashboard	5.1.3	6.1.11	7.1.5	T8

Table 8-3- Updated RTM after Testing

CHAPTER 9

RESULTS AND DISCUSSION

9. Results and Discussion

This section describes the results we obtained by following the approach in Section 7 for the high-level design described in Section 5.

We discuss results obtained for each classifier separately, followed by an analysis of the accuracy of our combined model, and scope for further improvement.

9.1. Knowledge Classifier

We discuss the results for the document classifier first, because this is critical to the second part of the Knowledge classifier.

9.1.1. Stage 1 - Document Classification

Subject	Accuracy in %
ADA	94
OS	93.5

Table 9-1- Accuracy for chapter classification

We attempted two approaches to train the Multinomial Naïve Bayes document classifier. The first approach involved splitting the text into individual paragraphs and sentences, and training a single classifier on the text. The label for the sentences is the section that that paragraph or sentence belongs to. This model gave us an initial accuracy of 45%, so we had to find a way to boost our accuracy.

This led us to our second method: train two levels of classifiers. The first level, Level 1, would be concerned with predicting only the **chapter** the sentence belongs to. The second level, Level 2, would be concerned with predicting the **section** for that particular chapter predicted by the Level 1 classifier. This allowed us to attain the accuracy shown in table 9.1.

The reason we can attain such a high level of precision is due to the nature of our

classifier, the Naïve Bayes Classifier, which is adroit at such simple tasks involving blatant keyword match based predictions. All this classification boils down to, is prediction of the right class based on the keywords in the input.

9.1.2. Stage 2 - Knowledge Level Prediction

Subject	Accuracy in %
ADA	55-70
OS	60-65

Table 9-2- Accuracy along knowledge dimension

Out of the 3 semantic models that we experimented with, LDA was consistently the best performer. The accuracies indicated in table 9.2 are those that were obtained by passing LDA probabilities along with the confidence value of the document classifier as input to a multi-layer perceptron (artificial neural network). We also tried passing LSA, Doc2Vec, and combinations of the 3, but they were all a few % points under LDA.

The reason for such a large fluctuation in accuracies is due to the unsupervised nature of the semantic models. Depending on how these models are trained, the features that are learned and given importance to vary greatly. There is no formal tuning theory to provide any guidance as to which is the best combination of learning parameters – there is no one-combination-fits-all either. The best combination is achieved through trial and error and varies greatly with the input that is passed.

In conclusion, it is not possible to formally benchmark and evaluate the soundness of our results, as this experiment has never been performed before. The only way to validate our system is to perform human recall.

9.2. Cognitive Classifier

In this section, we discuss in turn, each of the 3 models used to perform classification on the Cognitive dimension, and then analyse the performance of an ensemble built upon these 3 pre-trained models. The USP of this model is that it is not constrained to any subject, rather it is a generalised classifier that can classify any question based on the keywords observed in the body of the question. As documented in the algorithms in Section 7.1.3, questions undergo a substantial amount of pre-processing to bring them to a state where they can be accurately classified by our models, the most notable of which is the TFIDF threshold based filtering while retaining words if present in our keyword bank.

9.2.1. Multinomial Naïve Bayesian Classifier

Subject	Accuracy in %
<Any subject>	86

Table 9-3- Accuracy for MNBC along cognitive dimension

Why MNBC excels at this is again, due to its ability to match keywords with impeccable precision.

9.2.2. SVM - GloVe Classifier

Subject	GloVe Vector Size	Accuracy in %
<Any subject>	100	83
<Any subject>	300	78

Table 9-4- Accuracy for SVM - GloVe along cognitive dimension

This Classifier has been trained on generic GloVe vectors. Each word in the filtered input has been converted to its 100/300-dimensional GloVe equivalent. These words are also assigned an IDF weightage based on the meaning they convey to the structure of a sentence. Naturally, keywords are given the highest weightage. Each GloVe vector multiplied by its word weightage is then summed and the average taken over to get the actual input to the classifier. This classifier performs surprisingly well given the loss of information on taking the mean. We can also infer an interesting fact from table 9.4 above – the *degree of* loss of information in a 100-dimensional classifier is lesser than

in the 300-dimensional equivalent, and hence the 100-dimensional classifier does better.

9.2.3. Bidirectional Recurrent Neural Network

Subject	GloVe Vector Size	Accuracy in %
<Any subject>	100	76%
<Any subject>	300	82%

Table 9-5- Accuracy for BiRNN along cognitive dimension

This classifier is special because we implemented it in two ways – one was using keras and the other was by manually coding the mathematics behind the backpropagation through time algorithm. This was challenging but extremely rewarding since our handmade classifier could outperform its flimsy framework equivalent by a mile.

The only reason this model did not outperform any of the other cognitive classification models is simply the lack of data. Besides this, it does not have any of the flaws of the other classifiers – it does not suffer from information loss (because each word is handled in its own time step) and no independence assumption is made (as in the MNBC). There is enormous potential and scope for improvement. This is true for all our classifiers.

9.2.4. Ensemble Classifier

Subject	Method Used	Accuracy in %
<Any subject>	MLP based	88
<Any subject>	Weighted Voter	94

Table 9-6- Accuracy for ensemble model along cognitive dimension

Two methods have been employed in the construction of the ensemble. The first is the MLP (ANN) classifier which takes a set of probability distributions from each of our 3 pre-trained models, concatenated together to form an 18-dimensional input. The output is the final Cognitive level prediction.

The second method is one that assigns weights to each model based on its ability and confidence in predicting the right output. This model does not require training, rather, the manual tuning of weights. This, when done right results in an appreciable increase in the final accuracy. The ratio of weights is 2:3:1 for MNBC: SVM: BRNN.

CHAPTER 10

RETROSPECTIVE

10. Retrospective

Looking back on this project, much can be said about our 4-month experience working with our guide on it, enough so to be able to fill 60 pages on its own!

In a nutshell, here's what went right:

1. Each member of the team coming together to work on a ground-breaking project in a domain that was alien to us – what we call team synergy.
2. Bringing the project to an acceptable degree of completion before the end of the semester that will serve as a launching pad for future work in this area.
3. Learning about new concepts in machine learning
4. Learning how to work with tools, and learning how to understand when there is no point working with certain tools
5. Learning how to estimate time and effort
6. Improving our coding abilities beyond that of an academia level. This project has resulted in over 8000 lines of Python and Java code in just a span of 3 months – quite a feat for such a short time, going based on our past experiences with projects.

And here's what did not go well:

1. The crushing feeling of despair upon encountering an insurmountable roadblock – this is something that takes time to recover from.
2. Having to face bitter disappointment when we did not get positive results, and having to report the same to our guide.

Overall, it was a solid learning experience, and all factors considered, we've become much stronger at the end of it.

CHAPTER 11

REFERENCES

11. References

[1] Bloom, Engelhart, Furst, Hill and Krathwohl, "Taxonomy of Educational Objectives, Handbook I: The Cognitive Domain," 1956.

[2] Anderson, Krathwohl, Airasian, Cruikshank, Mayer, Pintrich, Raths and Wittrock, "A Taxonomy for Learning, Teaching, and Assessing: A revision of Bloom's Taxonomy of Educational Objectives," 2001.

[3] "Stack Exchange API," [Online]. Available: https://api.stackexchange.com/.

[4] J. Pennington, R. Socher and C. D. Manning, " GloVe: Global Vectors for Word Representation".

[5] N. V. Pujari and Nagashree, "A Tutor Assisting Novel Electronic Framework for Qualitative Analysis of a Question Bank".

[6] N. V. Pujari, B. HS and G. Akalwadi, "Application of Bloom's Taxonomy in day to day Examinations".

[7] Abdulhadi, Jabbar and Omar, "Exam Questions Classification Based on Bloom's Taxonomy Cognitive Level using Classifiers Combination".

[8] Osman and Yahya, "Automatic Classification of Questions into Bloom's Cognitive Levels using Support Vector Machines".

[9] Pincay and Ochoa, "Automatic Classification of Answers to Discussion Forums According to the Cognitive Domain of Bloom's Taxonomy using Text Mining and a Bayesian Classifier".

[10] A. Osman and A. A. Yahya, "Classifications of Exam Questions using Linguistically-Motivated Features: A Case Study Based on Bloom's Taxonomy".

[11] D. A. Abduljabbar, G. Al-Khafaji and N. Omar, "Programming Exam Questions Classification Based On Bloom's Taxonomy Using Grammatical Rules".

[12] Omar, S. S. Haris and Nazlia, "Bloom's Taxonomy Question Categorization Using Rules And N-Gram Approach".

CHAPTER 12

USER MANUAL

12. User Manual

This section details the use of the system as a tool for determining question quality, as well as using the system to analyse the performance of students in a test. Keeping in mind the possibility that this may be continued upon by future students, we have added a very detailed and informative set of step by step instructions to ensure that anyone can dive right into the code after reading this manual.

To get started, either burn a copy of the source code onto your machine from the CD. Once the GitHub repository is made public, you can clone the even clone the code at the repo URL using:

```
$ git clone https://github.com/doodhwala/question-quality-analyser
```

Also, ensure you have xpdf and mupdf tool installed on your machine:

1. The xpdf tool for parsing pdfs can be downloaded from their website, http://www.foolabs.com/xpdf/download.html
2. To download mupdf, either download the distribution from the website at https://mupdf.com/, or if you are using a Mac OSX machine, you can install it with:
   `$ brew install mupdf.`

Once done with these steps, ensure that you have downloaded all the required software listed as dependencies in the Section 3.2.2. Then you will be able to run all our scripts and classify questions.

12.1. Adding a New Textbook into the System

This section details what exactly needs to be done to add a new textbook to the system. Once the textbook has become a part of the system, one may then use one of our visualization tools to interactively feed in questions and get a classification according to Bloom's grid.

1. Place the PDF of the textbook that you want to want to add into the system in the *resources* directory. For more information on the codebase structure of the project, refer to Section 7.2. Make sure the PDF name is something small and easy to remember, like *CN.pdf*. In this case, the tag <Your subject> used in subsequent steps will refer to *CN*.

2. In the *preprocessing_code* folder, study the pdfparser_os.py file carefully. You will need to write some very similar code to do the same for your new pdf. Make changes in all the necessary places. Create a new file if needed, called pdfparser_<Your subject>.py and run it to generate section wise text files inside the *resources* subdirectory,

   ```
   $ python pdfparser_<Your subject>.py
   ```

3. Once this is done, you will need data to train the models. For this, you will need to get at least 150-200 questions of varying difficulty from the subject domain. Use your judgement to select questions of varying difficulty, and do ensure that the difficulty level is distributed across the taxonomic grid for best results during training. Hand label this data and save it as a csv file called <Your subject>.csv inside the *datasets* subdirectory.

4. Run qfilter_train.py on the *resources/<Your subject>* directory to generate a TFIDF model that will be used for filtering later. The command for this is,

   ```
   $ python qfilter_train.py <Your subject>
   ```

If you did the above steps without an error, then a TFIDF model should be created and saved inside the *models* folder.

5. Inside utils.py, there is a function called get_data_for_cognitive_classifiers(…). This function currently supports loading data for two subjects and one generic dataset. You'll need to write some code to load your own data as well, from <Your subject>.csv. Examine the code for loading the data from the csv files, and replicate it.

6. Repeat step 5, this time for get_data_for_knowledge_classifiers(…), also in utils.py

7. One by one, run all of the cognitive classifier scripts:

```
$ python brnn.py
$ python mnbc.py
$ python svm_glove.py
$ python cogvoter.py
```

Specifying a subject is not necessary, but you can modify what is trained on, by specifying the subject in the *what_type* parameter of the get_data_for_cognitive_classifiers(…) function call in all the above scripts.

8. Run the knowledge classifier programs for your subject,

```
$ python3 nsquared.py <Your subject>
$ python3 nsquared_v2.py <Your subject>
```

After carrying out steps 1-8, the models have now been created and saved in the *models* subdirectory, and are available for use by the visualization tool. You can now move onto section 12.2.

12.2. Visualization with the System

After downloading all the dependencies mentioned in Section 3.2.2, run the following command from within the `Code` folder to view the UI:

```
$ python3 visualize.py <Your subject>
```

<Your subject> could be either *ADA*, *OS*, or any other custom subject you created by referring to the guide in Section 12.1. Type a question in the input box and click the button.

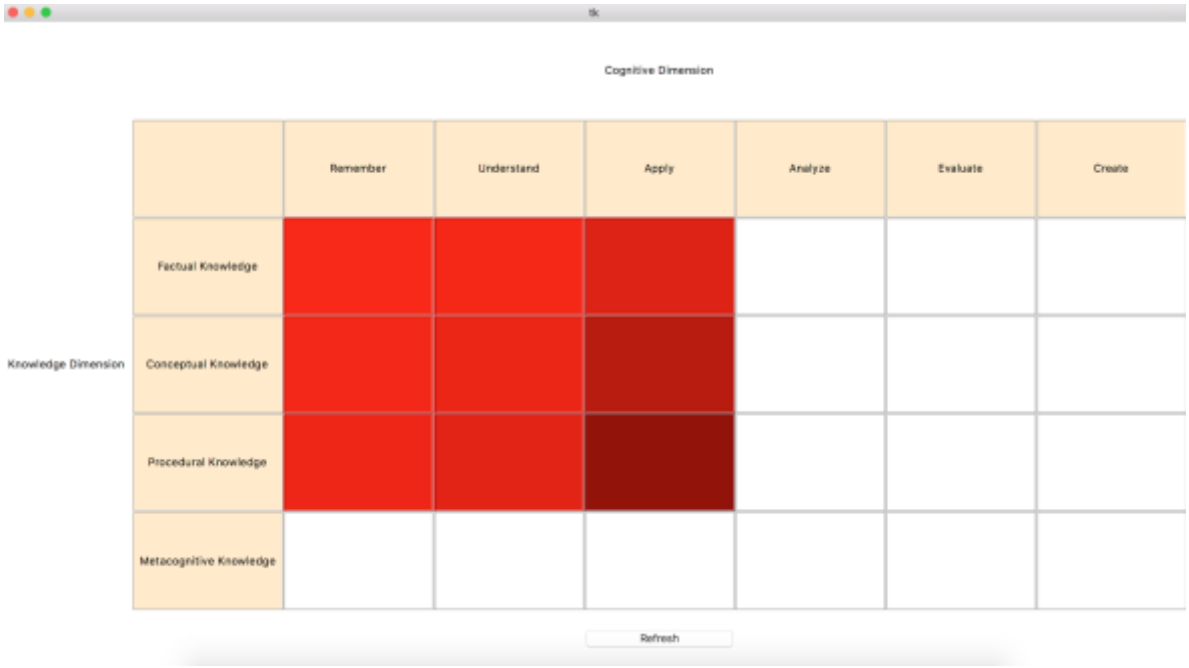

Figure 12-1- Heatmap generated for a particular question in the tkinter UI

A shaded grid will be presented to you that indicates the Bloom's knowledge and cognitive levels for the typed question.

12.3. Analysis of Student Performance in an Exam

Go to the `Web-UX/` directory, and run the following command from within the folder to view the Django dashboard:

```
$ python3 manage.py runserver
```

Open your web browser and navigate to:

```
http:// localhost:8000/reports/
```

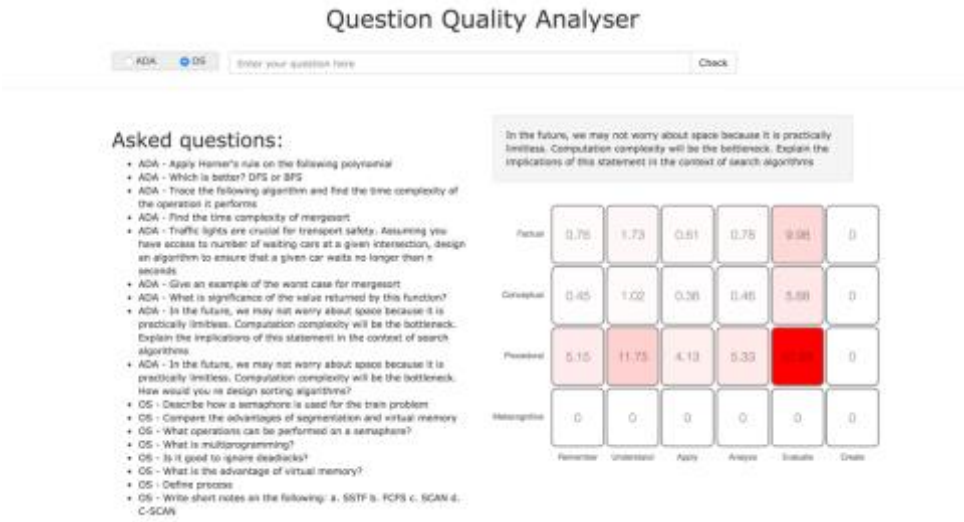

Figure 12-2- The heatmap generated for a particular question in the Django UI

Here, you can pick the subject using the radio button and type a question. When you click "Check", the question is added to the left side and the heatmap on the right side is updated with the scores for classification across the two dimensions. Past question values are cached, and hence, clicking on any question will display the scores for those questions instantly.

Another feature is the analysis of classroom performance. Once the question paper and student scores have been uploaded as a CSV file, the analysis of the performance is found at:

```
http:// localhost:8000/reports/analysis
```

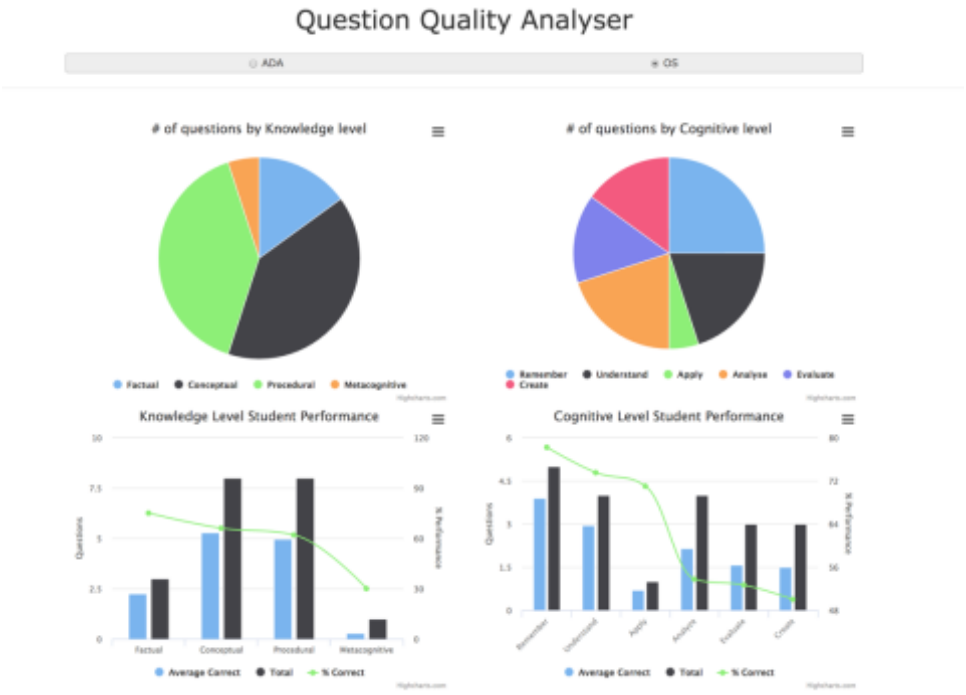

Figure 12-3- The teacher's dashboard generated for a question paper and class of students.

The pie charts on top represent the distribution of the questions asked based on the knowledge and the cognitive levels.

The bar graphs on the bottom represent the average correctly answered questions (light blue) vs the total questions asked (dark blue) per difficulty level.

The green line graph quantifies the performance per difficulty level by providing us with the percentage value for the average correctly answered by the total number of questions asked.