Web App for Quiz Generation

Abdul Mannan
dept. of Computer Science and Engineering
PES University
Bangalore, India
kanjimannan@gmail.com

Ishita Chaudhary
dept. of Computer Science and Engineering
PES University
Bangalore, India
ishita1001@gmail.com

Jason Carvalho
dept. of Computer Science and Engineering
PES University
Bangalore, India
jasonjcarvalho@gmail.com

***Abstract*— In this paper, we describe a tool we created to help students with revision. Our project is a web application which takes in a PDF as an input and then converts the pdf to text. Based on this text, the user gets an automatically generated quiz which consists of two types of questions. The user is then given a report containing the correct answers. The project is divided into 3 main parts - PDF to text conversion, Automatic Question Generation and Building the UI(Web application).**

Keywords— PDF to text conversion, Automatic Question Generation, Building the UI

I. Introduction

An integral part of studying effectively, and one students often struggle with, is revision. An effective way to learn is to constantly quickly quiz yourself on individual topics while studying and revising. Thalheimer (2003) [1] explains some of the benefits of using questions to learn, such as :
1) helping students realise their misconceptions about a particular topic 2) focusing their attention on the important study materials; 3) reinforcing learning by repeating core concepts; and 4) providing motivation to students to participate in learning activities that benefit them.
However necessary, constantly coming up with tests and questions is a cumbersome process. With the increasing role of computers in education, the interest in Automatic Question Generation systems is also increasing exponentially. While there are still a lot of issues regarding the standard and usability of questions generated by AQG systems, one cannot deny their usefulness in facilitating quick learning. The aim of our project is to automatically generate questions and present it to the student in the form of a quiz. The quiz would be based on a PDF file which the student uploads on the website. Our question generator outputs two types of questions- fill-in-the-blank questions and MCQs.This quiz would help the student to test her/his knowledge of the topics present on the page. This would also help students in revising effectively. As we are using automatically generated questions, it would reduce the workload of the teachers and be useful for self-study.

II. Literature Review

While extracting text out of PDF documents accurately is not an easy task, there have been several developments and tools built over the years which give excellent results. One such tool is the 'PDFminer'[9]. This parses all objects of a PDF document into Python objects. The layout analysis consists of three different stages: it groups characters into words and lines, then it groups lines into boxes and finally it groups textboxes hierarchically. The resulting output of the layout analysis is an ordered hierarchy of layout objects on a PDF page. Hence text is analysed and grouped in a human-readable way.

PyPDF2[10] is a python PDF library capable of splitting, merging together, cropping, and transforming the pages of PDF files. It can also add custom data, viewing options, and passwords to PDF files. It can retrieve text and metadata from PDFs as well as merge entire files together.

Using this library we can use the various methods present in it to extract the text from pdf files. For example the .numPages method would generate the total number of pages present in the pdf. The getPage(page_num) method returns a single page of the file as a Page Object and the extract text method gets the text from the text file in a String format. Hence by using the above methods we can extract the text from a pdf to a text file.

Automatic question generation has come out as a promising area of research in the field of Natural Language Processing (NLP) due to its wide applications in Educational Technology. Two major parts of this process is Sentence Extraction and Keyword Selection.

Mihalcea, Rada, and Paul Tarau [2] present TextRank is a graph-based ranking model mainly utilised for keyword and sentence extraction. This model basically decides the importance of a vertex in a graph based on a 'voting' or 'recommendation' system. For sentence extraction, this vertex is a sentence. A text unit recommends other related text units, and the strength of the recommendation is recursively computed based on the importance of the units making the recommendation. If a sentence has similarities (calculated by using the number of common tokens of the sentences) to another sentence, it gets upvoted by that sentence. Hence, the sentences are ranked in the order of their votes. Evaluation using the rouge method and based on guidelines by DUC evaluators showed that TextRank succeeded in identifying the most important sentences. The element that makes TextRank unique is that it is unsupervised, and only depends on the provided text to produce the key sentences.

Dhaval Swali,Jay Palan and Ishita Shah[3] have created in 2016, a question generator that uses a mix of syntax and semantic based approach to generate wh- type questions from both simple and complex sentences. It makes use of the Stanford Tagger for POS tagging. It then uses NER over the sentence for identifying the Subject, Object, etc. Questions on complex sentences are determined by the

discourse connective(conjunctions). It also extracts the auxiliary verb (if present) to help figure out the tense of the sentence, which is important in proper question generation.

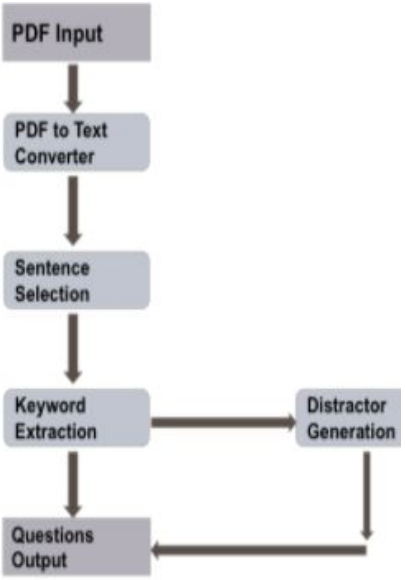
An interesting technique developed was to generate fill-in-the-blank questions on informative sentences in the text data by Das and Majumder [4]. The method identifies and selects these sentences based on Part-of-Speech tags and other rules. It does not follow a template based method. They also generate hints to nudge the user in the direction of the answer. The questions which were answered by identifying domain specific words from the text (an ontology approach) were removed from the review. The accuracy of the system for identifying informative, factual sentences was 92.367%.

'Ontology-Based Multiple Choice Question Generation', Alsubait [5] explains that the most difficult part of MCQ generation is generating distractors. They describe an ontology based approach to generate distractors. They have developed a 'Similarity Measurer', inspired by Jaccard's similarity coefficient, to generate questions of varying difficulty by varying the similarity between the key and distractors. This also ensures that the distractors created belong to the domain of the generated question. Two ontologies have been used for testing, and a question was deemed too difficult for students if it was answered correctly by less than 30 % of the students and was too easy if answered by more than 90 % of the students. External reviewers also evaluated if the generated questions were useful or not, with 92% and 96% declared as useful by at least one reviewer of each of the two ontologies used.

Rapid Automatic Keyword Extraction (RAKE)[6] is an unsupervised method for extracting keywords from text data. Unlike most of the work done in this area, RAKE is domain-independent. RAKE uses stop words and phrase delimiters to partition the document text into candidate keywords. Word Co-occurrence is identified within these candidate keywords. After every candidate keyword is identified and the graph of word co-occurrences is made, a score is calculated for each candidate keyword and defined as the sum of its member word scores. The degree and frequency of word vertices in the graph are used to determine the word scores. This model selects one third of the top scoring words in the graph as keywords. RAKE effectively extracts keywords and outperforms the current state of the art models in terms of precision, efficiency, and simplicity. As the number of words in the document increases, RAKE outperforms TextRank.

III. METHODOLOGY

To the web application, the user uploads the study material in the form of a PDF. We then convert this PDF and store the contents in a text file. This text file is passed as input to the question generator module, which generates fill-in-the blank(FIB) questions and MCQs. The user is then redirected to the Quiz page. After they're done, they click the submit button and the answers to the quiz are displayed on another page. All these components are modular, and integrated together using flask. The detailed procedure is as follows.



1. Flowchart of the procedure

A. PDF to text conversion

For the user to upload the PDF, Flask-which we are using for the backend of the website- provides a simple method. We use a simple form with the input type specified as 'file' for that submit button. The file is then taken from the form data and sent as data to the ('/convpdf') route, using an AJAX call. We then use this as the parameter for the PDF to text converter function. After experimenting with several PDF-to-text converters, we found that 'PDFminer.six' gave us the best results along with ease of use. Hence, we used this in the PDF to converter function. Pdfminer.six extracts the text from a page directly from the source code of the PDF. It is capable of copying the format of the text very accurately. You can also extract text, images, table-of-contents, tagged contents, etc. It is built in a modular way such that each component of pdfminer.six can be replaced easily with our own module if need be. It parses PDF objects into python objects and performs an analysis of the layout.

The layout analysis consists of three different stages: it groups characters into words and lines, then it groups lines into boxes and finally it groups text boxes hierarchically and thus finally forms the page. The output of the layout analysis heavily depends on a couple of parameters.

To group characters into words and lines, pdfminer.six uses the x and y coordinates that are present on the upper-right and bottom-left of each character, which is referred to as its 'bounding box'. Characters that are both horizontally as well as vertically close to each other are grouped on one line and how close they are to each other is determined by the char_margin and the line_overlap parameters. The char_margin parameter is relative to the maximum width of either one of the bounding boxes whereas the line_overlap is relative to the minimum height of either one of the bounding boxes. A space is inserted if the characters are further apart than the word_margin. The word_margin is relative to the maximum width or height of the new character.

Each line has a bounding box that is determined by the bounding boxes of the characters that it contains. Just like in the grouping of characters, pdfminer.six uses these bounding boxes to group the lines. Lines that are both horizontally overlapping and vertically close are grouped. How vertically close the lines should be is determined by the line_margin parameter. This margin is specified relative to the height of the bounding box.

The last step is to group the text boxes. This step recursively combines the two text boxes that are the closest to each other. The closeness of the bounding boxes is computed as the area that is between the two text boxes, i.e, it is the area of the bounding box that surrounds both the lines minus the area of the bounding boxes of individual lines.
Thus, the PDF is now converted into text, which we store into a text file using file IO operations.

B. Automatic Question Generation

An important part of question generation is choosing the right sentences for generating the questions. Thus, accurate summarization of the content is the key to getting the most informative questions. We use TextRank, an extractive unsupervised summarization technique to generate our summary.

We first split the text into multiple sentences by tokenizing it and the sentences are cleaned to remove ambiguous characters and stop words using the stop-words provided by the NLTK module. Next, we find the vector representation of every sentence in the text. We use GloVe word embeddings which are the vector representation of words. The pre-trained Wikipedia 2014+ Gigaword 5 GloVe vectors are used for our project. This has about 400000 vectors for words in the dictionary. Words that have similar meanings are vectors closer to each other.

To calculate vectors for our sentence, we take vectors for the words in the sentence and then the mean of those vectors is taken to get a single vector representing the sentence. Then we use cosine similarity to find the relation between the sentence vectors we just made and is stored in a matrix of size n*n (n is number of sentences). This similarity matrix is then converted to a graph where the sentences are the vertices and the similarity score of each sentence is the edge. The PageRank algorithm is then run over the graph to obtain the ranks. The top few ranked sentences are then selected to make the summary.

We use the Multipartite Rank algorithm from the pke[8] module to get keywords from the whole text. In this model, first a graph is built from the document. The ranking algorithm then assigns each key phrase a relevance score. Key phrase candidates are selected from sequences of adjacent nouns with one or more preceding adjectives. The edges between the nodes (key phrases) represent the co-occurrence relation between them, usually controlled by the distance between the word occurrences. The weight between the nodes is the sum of the inverse distances between the occurrences of the nodes in the graph. Once the graph is built, the TextRank algorithm is used to obtain the score for each vertex. Initially set to 1, the ranking algorithm which takes into account the edge weights, is run for several iterations until it converges. Once the final scores are obtained, they are arranged in reverse order of their score and top vertices in the ranking are the extracted keywords.

We make sure that stop words or punctuations are not considered as keywords. The selected keywords are further filtered based on the condition whether they appear in the summarized text or not. These filtered keywords are then used to obtain the sentences to be used as questions.

For generating the distractors, we make use of WordNet[7] and ConceptNet API to populate our distractor list and then the MCQ's are generated. WordNet is a large lexical database of English words used for various computational linguistics and Natural Language Processing. Nouns, adjectives, verbs and adverbs are grouped into sets of synonyms representing a distinct concept. These sets are then linked using conceptual semantic and lexical relations to give a network of meaningfully related words and concepts.

ConceptNet API returns JSON objects which provide information about the requested word like objects that relate it to other such nodes. We use this API to add more distractor words into our distractor list since sometimes WordNet isn't able to find similar words for certain keywords.
Our question generation module is thus ready.

C. Building the UI

With our pdf conversion and question generation modules ready, we need to just make them accessible from the website. The front end was created using HTML and CSS. The backend server was created using Flask, as our code is written in python.

We have created different routes for each module and different buttons on the page are used for calling these specific routes. When the user clicks on a certain button, an AJAX call to that route is made. The module(function) for that route is then called and the result is sent back.

When the user uploads the pdf, a call to the '/convpdf' route is made. This is a 'POST' request containing the PDF file. The function to generate text is called for this PDF, and the result is stored in a text file. Similarly, the button to 'Generate Questions' stores the data containing question and answer pairs in a global variable this time.

This data is then sent to ('/quiz') route for displaying the quiz, and then displaying the correct answers using separate templates. Here, we use templates because we need placeholders to display the dynamic data obtained from the Question Generation module. Flask uses the Jinja template library to render templates. Finally, the user can self evaluate their quiz results.

IV. Deployment and Testing

For testing, we used a PDF which contains an essay about Nature as our input PDF. It had about 80 sentences. After conversion to text and running the question generation on this text we obtain 3 MCQ questions and 3 Fill-in-the-Blank

questions. The number of questions generated depends on the size of the text. Given below are two examples of the questions that were generated:
Q. Initially, the _____ was not sustainable for any kind of living.
Answer: Earth.

Q. By taking small steps like planting trees, using biodegradable materials, stopping water pollution and keeping our surroundings clean, we can help _____ Nature breathe again.

- Father
- Parent
- Mother
- Earth

Answer: Mother

V. Future work and improvements

There are multiple areas which has some potential for some optimizations:

- Generation of wh-type questions- This involves a lot more knowledge in NLP to implement, involving POS-tagging and numerous sentence-question templates.
- A more interactive front-end for a better user experience.
- A model to make abstractive text summarization to grasp the complete sense of the document using LSTM's (Long Short Term Memory) and RNN's (Recursive Neural Networks).

VI. Conclusion

In this paper we present the review to automatically generate a quiz from a given PDF. As discussed above, many algorithms are required to make this Quiz Generator work. We use Pdfminer.six as well as PyPdf2 to generate a text file which is passed to the Quiz Generation part which undergoes the various processes and algorithms(Multipartite, TextRank, PageRank, etc) to give us the questions. Finally we build the web application which runs this quiz.
This quiz generator would help students, who are going to be the primary users, to revise and better prepare themselves for tests. Automatic question generation is an open area where there is scope of research in proposing methodologies by identifying complexities and types of questions like MCQs and Fill in the Blanks. While it may not be perfect, it will certainly save students a lot of time while preparing and hence be helpful to them. This is also helpful to the teachers as it would reduce their workload instead of having to constantly come up with new quizzes for students.

VII. Acknowledgment

This paper is primarily supported by PES University, Bangalore. We would like to thank Dr. S. Natarajan for mentoring us and guiding us through the course of this entire project.

VIII. References

[1] Thalheimer, W. (2003). The learning benefits of questions. Tech. rep., Work Learning Research.
[2] Mihalcea, Rada, and Paul Tarau. "Textrank: Bringing order into text." Proceedings of the 2004 conference on empirical methods in natural language processing(2004).
[3] Dhaval Swali,Jay Palan, Ishita Shah, Automatic Question Generation from Paragraph International Journal of Advance Engineering and Research Development Volume 3, Issue 12, December. (2016).
[4] Das, B., Majumder, M. Factual open cloze question generation for assessment of learner's knowledge. Int J Educ Technol High Educ 14, 24. (2017).
[5] Alsubait, T., Parsia, B. & Sattler, U. Ontology-Based Multiple Choice Question Generation. Künstl Intell 30, 183–188 (2016)
[6] Rose, Stuart & Engel, Dave & Cramer, Nick & Cowley, Wendy. (2010). Automatic Keyword Extraction from Individual Documents. Journal of Text Mining: Applications and Theory.
[7] Princeton University "About WordNet." WordNet. Princeton University. (2010).
[8] https://boudinfl.github.io/pke/build/html/unsupervised.html
[9] https://pdfminersix.readthedocs.io/en/latest/index.html
[10] https://towardsdatascience.com/pdf-text-extraction-in-python