PES-Question Answering Chatbot

Tanya Mehrotra 01FB15ECS323
Varun Ranganathan 01FB15ECS339

1. ABSTRACT

The product aims at solving MCQs given in the format of the CBT System used in PES University. This can be done by training the model with the books of the required subjects, and then, feeding a question paper to get its solution. This system can be used to automate the checking of computer based tests, as well as ease the teachers during test reviews. It can also act as an alternative metric to measure the students' performance rather than a simple average.

2. PROBLEM STATEMENT

Design an interface that would predict the correct answer for a given set of questions. The questions are of Multiple Choice format, which can later be extended to subjective questions.

3. METHODOLOGY

- **Preprocessing**

The book is received in a PDF Format. The first step is to extract text from the PDF so that it can be processed. The code is executed as follows :

<path to pypy> PreprocessPDFs.py <list of directories containing the PDF(s)>

"pdftohtml" converter is used which breaks down the PDF into an XML format, where each line is contained in the tag <text></text>. The strings are extracted from the XML formed.

"pypy" is being used here to gain speed up in computation as the main clustering algorithm is a O(n2) algorithm.

- **Cleaning**

The text in the entire book is contained in one String. Sentences are extracted using NLTK "sent_tokenize". The sentences received, are now cleaned, by using Porter Stemmer which stems each word to get it to its basic form. The stopwords are however, not eliminated, so that we get a proper list of important words that comprise the text.

- **Clustering**

Clustering of text involves a lot of approaches. KMeans is considered to be a good clustering approach for text, but the drawback is that we need to define the number of clusters beforehand. Since this is a very big assumption, this technique was discarded. This drawback was overcomed by Density-based spatial clustering of applications with noise (DBSCAN) , however, this method resulted into a lot of outliers and didn't cluster relatable sentences. Therefore, a new clustering technique was devised which has a different distance metric.

The distance metric consists of two aspects : cosine distance between sentences, and the relative position of the sentences in the document. Since a question-answering interface, when searching for answers, will look for relatable sentences, it is beneficial if sentences that have close proximity are clustered together. The sentence similarity was found for two sentences at a time. First, their cosine distance was computed, and next, a "difference" parameter was evaluated by the given formula :

$$difference = 1 - (sentenceDistance / 10)$$
$$Where,\ sentenceDistance = abs(position\ of\ S1 - position\ of\ S2)$$

The total similarity was calculated as:

$$similarity = 0.5*cosine + 0.5*difference$$

By the above formula, it is seen that equal weightage is given to both the aspects.

A similarity threshold of 0.5 is declared which would decide how the two sentences will be clustered. If the sentence similarity exceeds the similarity threshold, and if one of the two sentences already exist in a cluster, the other sentence is added to it, otherwise, both the sentences are grouped together a form a new cluster.

The similarity matrix, having dimensions of (len(total sentences), len(total sentences)) is updated for the corresponding sentences by the "*similarity*" computed above.

- **Named Entity Propagation**

In order to make more sense out of the sentences, a new technique, called "*Named Entity Propagation*" was adopted to declare the entities present in each sentence. This involves two steps : Find the entities for each sentence using "ne_chunk" NLTK, and then combine it with that of the previous sentence.

Eg: This is a bus. It is red in colour.

The “it” in the second sentence, after named entity propagation, would know that the reference is to “the bus”. This helps remove ambiguity from the text while it is processed.

- **Extracting Important words**

All the words, that are present in the sentences, as well as the entities formed after named entity propagation, are stored as important words.

- **Creating Search Matrix**

A search matrix mapping clusters and important words is created, which determines what important words are contained in a cluster.

- **Testing**

Testing done by executing the command:
<path to python> AnswerQuestions.py <Question Paper File Name>

This program takes a text file with the question and options for that question, and this is converted to a JSON string. We assume JSON to be our entry to the main logic of the program, which will make it easier for officials handling this bot. JSON format also enables us to create an API for this service.

Once the JSON is read, the whole paper is divided into a list of dictionaries of questions, where each dictionary will contain fields such as ‘type’ which is either ‘MCQ’ or ‘SUB’ (for now); question, which contains the question string; and options, which is the list of strings of options.

Each question is tackled at a time. The question sentence is first cleaned, and the important words are extracted from it. Now, we create a one hot vector, where ‘1’ indicates the presence of the word at that position in a global list of words. Now the computed search matrix is multiplied with this question vector, to give us the jaccard similarity between every cluster and the query. We also wanted to incorporate some form of feedbacks, for which we use a weights vector which assigns weight to each cluster, so that more important clusters are chosen. Therefore, the jaccard similarity is then multiplied by its respective cluster weight, and the cluster with the highest rank is chosen.

The sentences in the cluster are then searched to find which sentence has the most important words in it, relating to the cluster. That sentence is found using simple addition of important words present into the weight of that word in the cluster based on the TF-IDF method. Then, the options are processed, and we compare the options to the cluster by checking how similar are the options with the sentences in the query, and how far is it from the most important sentence. This gives us a score, which is turned into probabilities. Argmax on these probabilities give us the option.

4. RESULTS

The results of this project are excellent, as the bot was able to answer 5 out of 6 questions correctly when we chose the subject to be Theory of Computation. The one question it got incorrect was an inference based question which would require deep learning tactics to solve. Instead, with a bigger corpus, the inference may be present and the correct answer would have been outputted by the bot.

5. IMPROVEMENTS

Improvements that can be issued are with respect to the preprocessing. Due to Python's inherent Global Interpreter Lock, it does not use up the full potential of the resources. We could try to improve the "train" time by using another compiler such as IronPython or Jython, or provide methods for creating processes and provide methods for inter process communication.