ACTA TECHNICA NAPOCENSIS
Electronics and Telecommunications

REST WEB APPLICATION FOR GENERATING QUESTIONS BASED ON A TEXT

Technical University of Cluj Napoca, Cluj Napoca

Abstract: The subject of this paper consists of the generation of questions based on a text. At the end of the paper, we want to achieve an algorithm that can generate questions based on a text input (for instance lectures from university, or parts from the books and so on). The idea behind this application was the need for a tool smart enough to create questions from education materials in order for a better and easier way of learning. The thesis is structured in three main chapters in which is presented the basic notions needed, the algorithm and the final chapter is dedicated for the application.

Keywords: *Natural Language Processing, Natural Language Understanding, Artificial Intelligence, AngularJS, Django*

I. INTRODUCTION

How would someone tell if you read this text or not? They could ask you to summarize the text or to compare it to other papers from the same genre or the same topic. They could also ask you to discuss the weaknesses and strength of the paper.

Of course, if you are a highly skilled and motivated reader, then you will not be checked if you retained necessary information from the text. However, this is not the case with all the readers. For instance, an elementary school teacher might ask his or her students basic questions because they are still dealing with understanding complicated texts.

We aim to create a system for generating questions that take as input a text, for instance, a web page or an encyclopedia article or even a text from a course book. After the input was received, it is processed, and by that, we get to parse it, understand it and extract every helpful information from the received text. The final step is the generation of useful questions to see how much from the topic was retained by the reader.

Generating such questions can be a time-consuming and effortful process. In this research, we work toward automating that process. In particular, we are going to focus on generating questions from individual texts.

We are going to focus on non-fictional texts that convey factual information rather than opinion. While personal essays and narratives texts would be an exciting topic, we leave these problems to further work, and that is because this thesis is focused on generating questions from teachers' materials (lectures or course supports).

II. FUNDAMENTALS

This section will give an introduction into regex expression, natural language processing, natural language understanding and the grammar behind questions in English.

Regular expressions are an algebraic notation for characterizing a set of strings, put in other words, regex is a sequence of characters that define a search pattern. These expressions are beneficial for searching in a text when we are looking for instance through a corpus a given model.

When searching through a text, the search can be designed to return every match on a line, if there are more than one, or it can be set to return only the first apparition of the given pattern. For the following example, we generally underline the exact part that matches the model and only the first match. The backslash character doesn't usually delimit a regular expression, but in the current thesis, they will be shown with a backslash for better readability.

Depending on the regular expression parser, some of them may recognize only subsets of the pattern, or to treat some exceptions slightly differently. [1]

Natural language processing (shorthand NLP) is a subfield of artificial intelligence and computer science. It focuses on how to program a computer to process and analyze large amounts of natural language data. NLU is the shorthand for natural language understanding and it is a subtopic of computer science and NLP, that deals with machine reading comprehension. [2]

Before we start explaining the NLP tools that we used to achieve the question generator, we first need to establish some grand rules. We have to define what we consider a word. Let's take as an example the following sentence: "Apple is looking at buying U.K. startup for $1 billion." How many words are in there? There are 11 works if we don't count the punctuation marks as words, and 12 if we count them. Depending on the case, the algorithm should or shouldn't count punctuation as words. In our case, we will have to take in consideration also the punctuation.

Tokenization in computer science is the process of converting a sequence of characters (in our case our input text) into a series of tokens (strings with an assigned, and thus identified meaning). The resulting tokens are then passed to another form of processing. The tokenization process can be considered a sub-task of parsing the input.

The tokenization process is composed of the following steps. First, the raw text is split on whitespace characters, similar to the text.split(' ') function. The tokenizer then processes the text from left to right, performing two checks on each substring:

1. Exception checks. For example, "don't" doesn't contain white spaces but actually, there are two words inside that token, which are "does" and "not" and our tokenizer will split that word into "do" and "n't", while "U.K." should always remain one token.

2. Prefix, suffix or infix check. At this point, the tokenizer will try to split by special characters like commas, periods, hyphens or quotes. We will consider prefix: character or characters at the beginning, e.g. $, (, ". Suffix: character or characters at the end, e.g. km,), ", !. Infixes are the character or characters in between, e.g. -, --, /, ...

The tokenizer that we are going to use is the one from spaCy [3]. Because it's tokenizer can split, complex and nested tokens like combinations of abbreviations and many punctuation marks. A complicated example of the previously explained steps appears in the figure 1.

Figure 1. This is an example complex tokenization.

Classifiers, when talking about Natural Language Processing, labels tokens with category labels or class labels. They actually try to predict the class of given data points. Typically, labels are represented with strings (such as “politics” or “sports”).

For creating the desired question generator, we use the NLTK classifiers. As it is mentioned in NLTK documentation [4], classification is the process of choosing the correct class label for a given input.

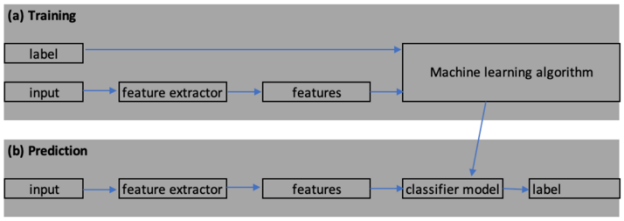

Figure 2. Supervised Classification.

In the above picture, during training ((a) Training), a feature extractor is used to convert each input to a feature set. For instance, the feature extractor for gender identification classifier would construct a dictionary that contains the last two characters of the word in order to figure out if the word is feminine or masculine, and that dictionary will be the features set. Pairs of features sets, and labels are then “fed” to the machine learning algorithm to generate a model (we will not go further talking about machine learning algorithms).

During prediction (stage (b)), the same feature extractor is used to convert unseen inputs to feature sets. This feature sets are then “fed” into the model, generated at the previous step, which generates predicted labels. [4]

A **Part-of-speech Tagger (short POS Tagger)** is a piece of software that takes as input some text in some language (in our case is English) and assigns parts of speech to each word (tokens), such as nouns, verb, pronoun, preposition, adverb, conjunction, participle and article.

The original POS Tagger was originally written by Kristina Toutanova. Since that time, Dan Klein, Christopher Manning, William Morgan, Anna Rafferty, Michel Gally, and John Bauer have improved its speed, performance, usability, and support for other languages, as it is specified in the documentation of the Stanford documentation [5].

Part of speech is a key feature in Natural Language Processing, because with the help of it we reveal a lot of useful information about the word and its neighbors. As it is presented in [2], knowing whether a word is a noun or a verb tells us about likely neighboring words (nouns are preceded by determiners and adjectives, verbs by nouns) and syntactic structure word (nouns are generally part of noun phase), making the POS tagger a key aspect of parsing. Parts of speech are also useful features for labeling named entities like people or organizations in information extraction.

As a POS tagger we are going to use the one from SpaCy, because it’s the best open source python library for Natural Language Processing. The framework uses terms like head and child to describe the words connected by a single arc in the dependency tree. The term dep is used for the arc label, which describe the type of syntactic relation that connects the child to the head.

Also using SpaCy build-in visualizer, we can see how each word is interconnected with the other words. This can be seen in Figure 3

Figure 3. Dependencies of the sentence in POS tagging.

III. QUESTIONS GENERATION

This chapter will mainly focus on the question generator. The first part presents an introduction to forming question in English, after that the algorithm that generates question will be presented.

Questions are the heart of understanding what we read. Asking questions is about interacting with the text to develop a profound understanding. In order to be capable to create a question generator, first we have to explain the types of question in English and also how they are formed. As it is presented in the article [6], in English we have 4 essential types of questions:

- Yes/No Questions
- “Wh” Questions
- Tag Questions
- Negative questions

In the thesis paper they are all presented in depth (how they are formed, when to use them and so on).

Next, we are going to present the algorithm for generation the questions. The algorithm is one combining the technique for the question generator from [7], [8] and [9]. The steps to achieve the factual question generator are:

1. Tokenization: Tokenize the input data received from the user using a tokenizer such as the one from SpaCy
2. POS-tagging: Assign part-of-speech to every token from the input using a POS tagger such as the one from Spacy or NLTK
3. True-casing determination: Assign the tokens their true capitalization case styling using a true-caser (also present in SpaCy framework or one such as Stanford CoreNLP's TrueCaseAnnotator). This step is especially useful for textbooks that include important vocabulary terms in a caps-lock style, but it could also be skipped if, for instance you are trying to achieve a question generator for newspaper articles.
4. Subject/phase determination: Build a list of subjects and corresponding phrases from the tokens and the part-of-speech tags using a multiclass classifier. A multiclass classifier or trivial determiner can then be used for question generation.
5. Synonym replacement. Convert some work into their synonym words. This step aims to prevents students or whoever is using the question generator from matching the words from the question to the ones in the input text, in order to answer the questions.
6. Additional information extraction: Associate additional information with each subject/phrase match that could be useful for question generation using regular expressions.
7. Question formation: Attempt to form a question of each question type or each subject/ phrase match using basic patterns of each question type as a guide. The types of questions and how they are formed is presented in the previous sub-chapter.
8. False answer formation: Form similar, but false, answers for each question based on the question type using patterns of good false answer for each question as a guide. For instance, true/false questions can be negated to determine the correct and incorrect answers, while word embedding or a system scoring similarity between words and POS tags of other subjects can be applied for fill-in-the-blank type question
9. Shuffle and return questions

After implementing all the steps from above you should have a question generator that is capable to create questions from a text input.

In order to achieve a better question generator, we will have to use the classifier that we spoke about (in chapter 1). When we receive the text as an input the first step will be to use the classifier to classify the text into a specific type. After classifying the text into a category, we have to mark the sentences that will be used to create questions. In order to figure out which sentence is in correlation with the subject of the text, a textual entailment classifier should be used. Textual entailment takes a text (the sentence in our case) and a hypothesis (the category class of the text) and checks if the two are in a relation. If they are, then we will mark the question for generating questions out of it.

IV CASE STUDY: StudHelper

Here we will go through the software development of the application, the problem that I ran into when designing the application and future improvements.

Let's start from the beginning; in order to have a fully functioning rest web site there are 2 components required: one is the API (the server side or the backend part of the program) and the client application (the frontend part). In order to understand everything, we will go through each part and briefly explain how those components work.

We mainly focus on the REST design of an API. REST stands for Representational State Transfer, and it relies on a stateless client-server, cacheable communication protocol – and in virtually all cases, the HTTP protocol is used. As a programming approach, REST is a lightweight alternative to Web Services and RPC.

The web-application is rather simplistic for mainly two reasons: first this is a website that should encourage the user to read and understand the text, we don't want to bother the user with pop-ups or commercials or anything like that; the second reason is that in this application we only want to show how the question generator algorithm works and to prove that it will helps people study faster.

The structure of our application was primarily designed in a Model-View-Controller manner. When developing the web application, we used AngularJS to design and develop the controllers and the views of the web application and the Django REST framework to design the models. In the figure 4 is presented the diagrams that illustrate the MVC design pattern that we used

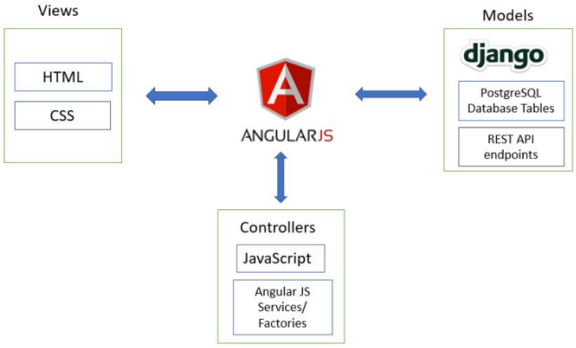

Figure 4. Diagram of the MVC design pattern.

The backend is written in Python. As a web framework we used Django [10]/Django REST framework [11] which is a powerful and flexible toolkit for building Web APIs.

If we are talking about the frontend application, it is done in AngularJS [12]. Angular is a framework for building client applications in HTML, TypeScript and CSS; it is written in TypeScript.

Next, I am going to present some figures of the application for a better understanding of what it is doing.

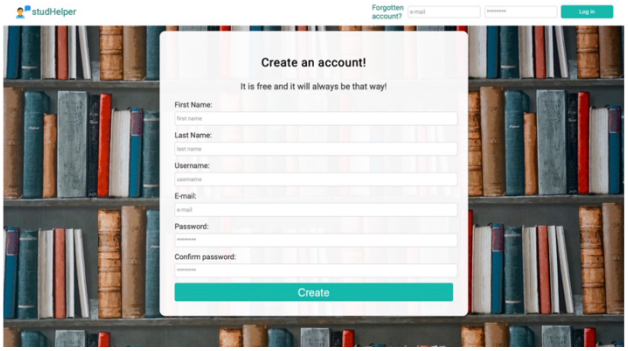

Figure 5. Homepage.

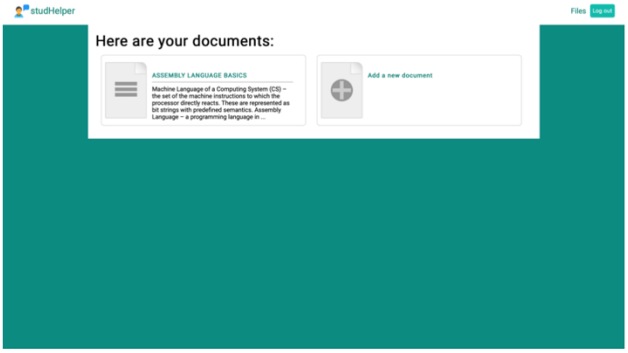

Figure 6. Homepage after you are logged in.

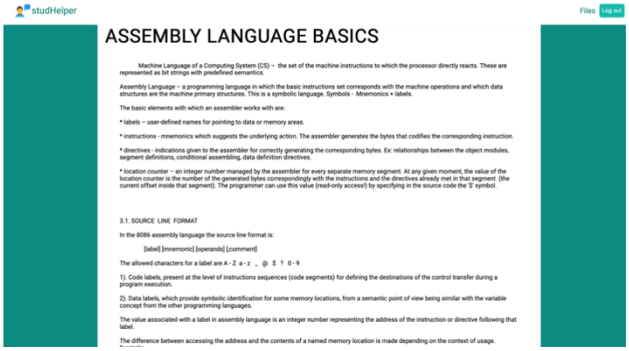

Figure 7. File page with the questions part 1.

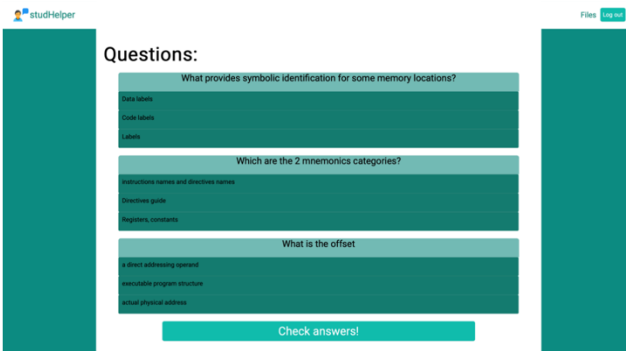

Figure 8 File page with the questions part 2. (The form isn't touched)

Figure 9. File page with the questions part 2. (After checking answers)

In this thesis, we presented the fundamentals of regex expression and natural language processing tools. We also showed how questions are formed in English and the steps to implement the algorithm that will generate questions. Experiments were conducted on various text inputs and data sets to make sure that the algorithm will be a useful one.

We believe that with the evolution in the field of artificial intelligence, a question generator could one day replace the way that people learn. Also by achieving the perfect algorithm one day the creation of tests and exams could also be made with the help of a question generator like this one, and this achievement would change the educational system in a better way.

REFERENCES

[1] J. H. M. Daniel Jurafsky, Speech and Language Processing.
[2] G. I. Schünz, Advanced natural language processing for improved prosody in text-to-speech synthesis, 2014.
[3] "spaCy Documentation," spaCy, [Online]. Available: https://spacy.io/usage.
[4] E. K. a. E. L. Steven Bird, "Natural Language Processing with Python - Analyzing Text with the Natural Language Toolkit," [Online]. Available: https://www.nltk.org/book/.
[5] "Stanford Log-linear Part-Of-Speech Tagger," [Online]. Available: https://nlp.stanford.edu/software/tagger.shtml.
[6] Kitlum, "The 5-step Guide to Forming Questions in English Grammar," [Online]. Available: https://www.fluentu.com/blog/english/questions-in-english-grammar/.
[7] M. Heilman, Automatic Factual Question Generation from Text.
[8] A. Kretch, "Question Generation: Using NLP to Solve "Inverse" Task," [Online]. Available: https://medium.com/@aleckretch/question-generation-using-nlp-to-solve-inverse-task-a92ff033bcf1.
[9] G. B. Y. Z. Xuchen Yao, Semantics-based Question Generation and Implementation.
[10] "Django Documentation," Django, [Online]. Available: https://docs.djangoproject.com/en/2.2/.
[11] MkDocs, "Django Rest framework documentation," [Online]. Available: https://www.django-rest- framework.org/api-guide/requests/.
[12] "Angular Documentation," Google, [Online]. Available: https://angular.io/docs.