

Automatic Multiple-Choice Question Generator by Using Machine Learning

Albertus Kelvin

School of Electrical Engineering and Informatics

Bandung Institute of Technology

13514100@std.stei.itb.ac.id

Abstract—English reading comprehension is one of the most required skill for people whose native language is not English. One example is when you study in an English spoken country, you have to read many literatures supporting the course in English language. Therefore, your ability to get the concept behind the literatures effectively determines your level of understanding towards the literatures. To achieve this, many universities or companies require the TOEFL, IELTS, or TOEIC score in order to know your basic English skill. Because of this, the applicants of university or company decide to take one of those exams and do many preparations by studying from any resources providing the practice questions, such as the English exam books or resources from the internet. Unfortunately, those resources provide the limited number of sample questions and the genre of article cannot be customized by the learners. This makes the learning process become not fun and not effective at once since the learners cannot set their preference in the literature’s genre and this makes they cannot improve their understanding in the other genres. This paper proposes an automatic multiple-choice question generator for English document. It runs on LINE platform as a bot so that the learners can interact directly and learn in more effective way. To achieve that, it uses LINE Messaging API to receive the user’s query and send the reply back to the user. The system does several things to generate the best questions, such as sets the blank position for the answer to every word in the document, determines the most reliable sentences (which has already had the blank position for the answer) to become the question, and generates the four options as the possible answers.

Index Terms—Automatic question generation, Levenshtein distance, LINE Messaging API, machine learning, multiple-choice question.

I. INTRODUCTION

ENGLISH reading comprehension is one of the most required skill for people whose native language is not English. Many university or company require the TOEFL, IELTS, or TOEIC exam result in order to know the applicant’s ability in several aspects of English language, such as reading, listening, writing, and speaking. From these aspects, reading comprehension sometimes becomes one of the four aspects

that needs a long enough time to complete since it requires a deep understanding on the concept of a certain article. What makes reading comprehension difficult is the complex story plot hiding the primary semantic of the article. Furthermore, the possible answers also provides many options having similar semantic and possibility to become as the correct answer. This makes the exam participants to go back to the first line of the article only to find the part of the article talking about the context asked in the question.

To achieve the best score in reading comprehension, we need to practice a lot. The common choices are learning from the internet and the books providing the practice questions. But, these methods have several downsides, such as the practice questions are limited and the learners cannot customize the genre of the article with their preference. For the latter reason, the learners cannot expand their knowledge in other genres even though it is a good thing to have many experiences in reading different genres so that they become more accustomed when reading the exam articles.

Therefore, we need a system that can generate high qualified questions automatically based on our own article. What we need to do is providing the article with the specified genre and the system will determine the important questions that can really examine your understanding towards the article.

In order to get the best learning process, the system will be integrated with LINE platform by using LINE Messaging API. The system which acts as a bot enables the users to learn in an interactive and fun way so the learning goal could be achieved more effectively.

II. RELATED WORK

There are several research papers written on multiple-choice question generation using machine learning approach. Ayako Hoshino and et al [1] state that machine learning approach can be used to generate question automatically since there are some processes that can be done using classification. They implement machine learning algorithms, such as Naïve Bayes and K-Nearest Neighbors to generate questions on English grammar and vocabulary from on-line news articles. They designed a system that can receive user input in the form of HTML file and turns it into a quiz session. The system does several things to generate the questions, such as extracting

features, deciding blank position, and choosing the distractors. Moreover, when the system receive the answer from the user, it shows the evaluation result, including an overall feedback.

Other than that, there is also a research done by Takuya Goto and et al [2] in which they built an automatic multiple-choice cloze questions generator from English text. They state that empirical knowledge is needed to produce appropriate questions, so they used machine learning approach to acquire knowledges from the questions. The knowledges are used to determine the most suitable aspects to be asked. The system does the following methods to produce multiple-choice questions: 1) extracts the appropriate sentences from the document that have high qualification to be considered as the questions using Preference Learning, 2) estimates a blank part based on Conditional Random Field, and 3) generates distractors based on statistical patterns of existing questions. This method of generation shows that it is possible to select the appropriate sentences and blank parts from the document as well as the possible answers (distractors). The method also works in the sentence that does not contain the proper noun.

III. METHODOLOGY

The method for finding the appropriate sentences uses machine learning algorithm in which the training data was taken from the collection of TOEIC exam questions. Before determining the sentences that can be considered as questions, each word in the sentences will be set as the blank position. Afterwards, the machine learning algorithm was implemented to determine which sentences with the blank position set before are the most suitable choices. Because of this approach, the training data consists the collection of questions with the original blank position. When the appropriate sentences have been determined, the next step was to generate the distractors. This was done by using one of the similarity algorithm, namely Levenshtein distance. The process of similarity checking was done by comparing the actual answer with all words in the English dictionary. The top four of the most similar words will be chosen as the distractors.

A. Preparing the training data

The training data is the collection of TOEIC past exam questions. The type of questions is multiple-choice with four possible answers and the exam taker must fill in the answer in the specified blank position. There are 10 categories of questions in which the total number of question for each category is 100. Each question has a format like this:

Register early if you would like to attend next Tuesday's ----- on project management.

a. seminar b. reason c. policy d. scene

The blank part ('-----') will be changed by [] in the training data just for simpler implementation. Moreover, the training data only contains the question and the distractors are omitted.

Other than that, the training data has two classes, namely

true (1) and *false* (0). The *true* labels indicates that the sentence is appropriate to be a question. Vice versa, the label *false* indicates that the sentence is not qualified to be considered as a question.

B. Determining the blank position for possible answers

Each word in a sentence in the training data is assumed to be a blank position. So, for this sentence *I have to go to school* will have 6 different representations in the training data. They are [*I have to go to school*, *I [] to go to school*, *I have [] go to school*, *I have to [] to school*, *I have to go [] school*, and *I have to go to []*]. Moreover, each representation has its own label which is *true* or *false*. Generally, a representation is classified as *true* when the word set as the blank position has significant meaning and represents the context of the article.

I use Naïve Bayes as the machine learning algorithm to build the model based on the training data. The model should be able to decide whether the new data is classified as *true* (1) or *false* (0) with certain probability degree.

C. Deciding the distractors

The system generates multiple-choice questions which means each question should have several options as the possible answers or distractors. In this case, the system has 4 distractors that will be used to assess the exam takers understanding towards an article.

Therefore, to really know the level of understanding of the exam takers, the distractors should provide possible answers having the similar meaning. In addition, when the exam takers choose one answer from the options, they should think deeper since each option has similar possibility to be considered as an answer.

One way to get the similar answers is by comparing the actual answer with each word in the English dictionary. However, the similarity algorithm implemented by this system does not consider the similarity in the context of semantic, yet only how far the word to be compared with each word in the dictionary. For example, the word RONALDINHO is similar with the word ROLANDO yet the semantic contained in each word is not similar.

Based on the chosen concept, this system uses Levenshtein distance algorithm to determine the word similarity. The algorithm computes the similarity of two words by computing the total amount of changes of letter to change one word to the desired word. The system iterates over the whole dictionary and compute the Levenshtein distance in each iteration. After the process is finished, the system takes four words in the dictionary which their Levenshtein distance states that they are similar with the actual answer.

D. Building the user interface

The users interact with the system through LINE platform which means they have to install LINE app so that they can use the service. The system will act as a bot where the general way of interaction is request and response. The users provide an article as the request and the system gives the questions as the response.

In addition, there are two ways in providing the article, namely by the URL of the article (online resource) or the original article which is copied and pasted to the message sender. For the first option, the system will do the crawling process to get the important part of the web page which means the system will try to find only the article in the page and not the others. For the last option, the system will do the analysis directly.

Since the system is integrated with LINE platform, the LINE Messaging API is used so that the request and response process can be done via LINE. The methods used for the process are push message and reply to the user. Both can be used for request and response event.

When the user wants to choose an answer, they do not need to write it down on the input textbox, yet the system provides four buttons in which each of them shows the possible answer. The user can directly choose an answer by clicking on the button then the answer will be sent to the server to be evaluated. The button also uses LINE Messaging feature, namely the ButtonsTemplate object.

Although the system acts as a bot, it can only receive valid requests from the user. Any invalid requests will be ignored and the response sent back to the user will state that their request cannot be evaluated. However, any text having more than 50 characters will be considered as the valid request and categorized as the article requested directly by the user (remember that the user have two options in providing their article). Besides of that, any text having valid URL format will be considered as the link to the requested article. To achieve that, the system also has the ability to determine whether a request is an URL or just an ordinary text.

IV. IMPLEMENTATION

In this part, I will show the result of the implementation of the method. Here is the sample of data used to train the classifier.

```
0, [] think the proposal represents the public needs
1, I [] the proposal represents the public needs
0, I think [] proposal represents the public needs
1, I think the [] represents the public needs
1, I think the proposal [] the public needs
0, I think the proposal represents [] public needs
1, I think the proposal represents the [] needs
1, I think the proposal represents the public []
0, [] need to study hard to get the best score
1, You [] to study hard to get the best score
0, You need [] study hard to get the best score
1, You need to [] hard to get the best score
1, You need to study [] to get the best score
0, You need to study hard [] get the best score
0, You need to study hard to [] the best score
0, You need to study hard to get [] best score
1, You need to study hard to get the [] score
1, You need to study hard to get the best []
```

Image 1. Sample of training data

As we can see, each word in a sentence is replaced by [] to indicate the possibility of blank position to be placed there. Each sentence is classified as *true* (1) or *false* (0) based on the importance of the position of possible answer.

The next step is training our Naïve Bayes classifier using the training data. The system used Naïve Bayes as the classifier and StringToWordVector as the filterer. Here is the result of training which was tested using 10-Cross validation.

Correctly Classified Instances	1288	74.2791 %
Incorrectly Classified Instances	446	25.7209 %
Kappa statistic	0.0097	
Mean absolute error	0.2825	
Root mean squared error	0.4609	
Relative absolute error	129.8351 %	
Root relative squared error	139.8368 %	
Total Number of Instances	1734	

=== Detailed Accuracy By Class ===

	TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
	0.821	0.809	0.878	0.821	0.848	0.010	0.454	0.859	0
	0.191	0.179	0.131	0.191	0.155	0.010	0.454	0.126	1
Weighted Avg.	0.743	0.731	0.785	0.785	0.743	0.762	0.010	0.454	0.768

=== Confusion Matrix ===

```
a  b <-- classified as
1247 272 | a=0
174 41 | b=1
```

Image 2. Model evaluation using 10-Cross validation

Also, here is the code used to build the model using Naïve Bayes classifier and StringToWordVector filter. The training data has already loaded beforehand.

```
/**
 * This method trains the classifier on the loaded dataset.
 */
public void learn() {

    try {

        trainData.setClassIndex(0);
        filter = new StringToWordVector();
        filter.setAttributeIndices("last");
        classifier = new FilteredClassifier();
        classifier.setFilter(filter);
        classifier.setClassifier(new NaiveBayes());
        classifier.buildClassifier(trainData);
        // Uncomment to see the classifier
        // System.out.println(classifier);
        System.out.println("===== Training on filtered (training) dataset done =====");

    } catch (Exception e) {
        System.out.println("Problem found when training");
    }

}
```

Image 3. Code for training the classifier

After the model is built, the system can receive requests from the users. The system offers two methods for the users in providing the request, namely through an URL that goes to a web page containing the article or just direct article which is copied and pasted to the input textbox.

Suppose the user give the URL of article then the request will be sent to the server to be analyzed. The response from the server states that there are 584 questions that are eligible to assess the user's understanding. Here is the screenshot.

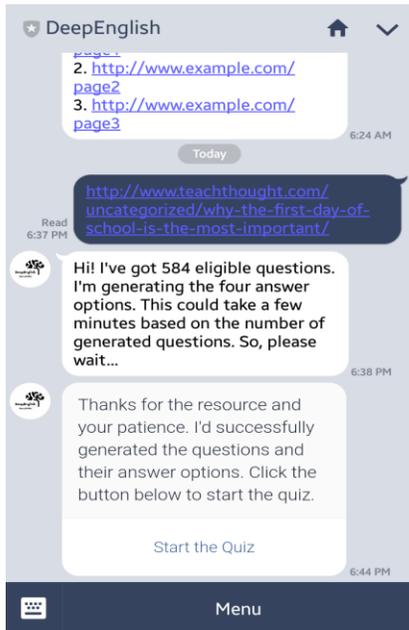


Image 4. The URL of the page containing the desired article and the response from the server

When the user click on the 'Start the Quiz' button, new request is sent to the server and the user will get the questions as the response from the server.

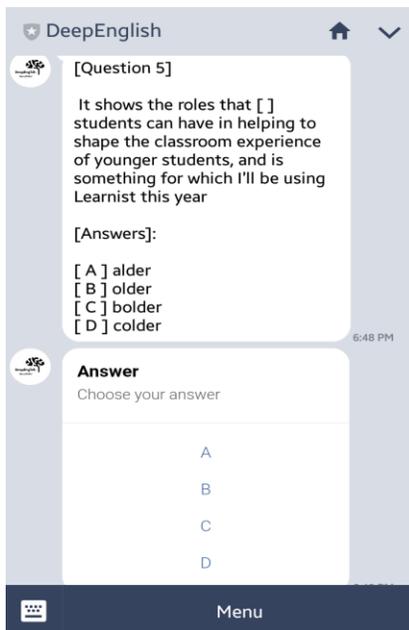


Image 5. Question number 5 with the ButtonTemplate containing the distractors

And here is another example of generated question sent from the server.

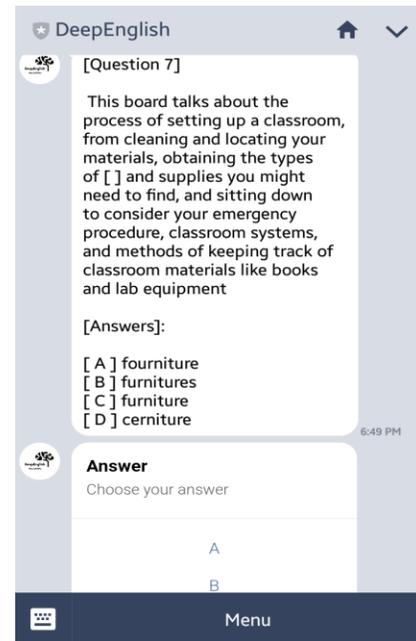


Image 6. Question number 7 with tricky options

V. CONCLUSION

When we need to ask a question, we should have knowledge on the essential concepts that exist in the text since it enables us to generate question related to those concepts. In this paper, I used machine learning approach to get those knowledge which is implemented in the process of deciding the blank position of the eligible sentences.

Based on the implemented methods, the system is able to generate questions from an article written in English language by doing several procedures, such as creating the same sentences with the same amount as the total number of words contained in the corresponding sentence, making each word in the generated sentences as the blank position, deciding the class of each sentence (1 or 0), retrieving all sentences having *true* (1) as the class, generating the distractors by using Levenshtein distance algorithm, and sending back the result to the user.

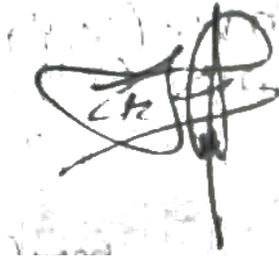
REFERENCES

- [1] Ayako Hoshino and Hiroshi Nakagawa. 2005. A realtime multiple-choice question generation for language testing: A preliminary study. In Proceedings of the ACL 2005 The Second Workshop on Building Educational Applications Using Natural Language Processing, to appear.
- [2] Goto, T., Kojiri, T., Watanabe, T., Iwata, T., & Yamada, T. (2010). "Automatic Generation System of Multiplechoice Cloze Questions and its Evaluation", Knowledge Management & E-Learning: An International Journal (KM&EL), Vol 2, No 3, 2010

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 5 Mei 2017

A handwritten signature in black ink, appearing to be 'Albertus Kelvin', written over a faint, circular stamp or watermark.

Albertus Kelvin / 13514100