Multiple Choice Question Generation for Slides

Ainuddin Faizan[1], Steffen Lohmann[2] and Vinay Modi[3]

[1] RWTH Aachen University, Germany
ainuddin.faizan@rwth-aachen.de
[2] Fraunhofer IAIS, Germany
Steffen.Lohmann@iais.fraunhofer.de
[3] University of Bonn, Germany
modi@cs.uni-bonn.de

Abstract

Be it a traditional classroom environment or a state-of-the-art e- learning platform, assessment, in the form of a series of questions, is almost always included. Questions provide the teacher or the learners themselves, a way to judge the extent of the learners knowledge. Multiple choice questions (MCQs) are one such type of questions that provide the learner with a question and four choices, one being the answer and the other three being distractors. This paper presents an approach to use semantic annotation in order to generate a variety of MCQs from slide content. Different strategies are discussed to use and manipulate data from the semantic web in order to generate questions and distractors of varying difficulty.

1 Introduction

E-learning platforms like SlideWiki[1] present information as slides, along with questions for assessment. These questions are manually created, which can be a tedious task.

We have developed an approach that provides assistance by generating three varieties of multiple choice questions (MCQs) automatically based on the content of the slides. Several researchers have generated questions from different sources like domain ontologies and knowledge bases [1, 2, 3]. We use semantic annotation to find DBpedia[2] resources in slides and then use SPARQL queries to get distractors. The lack of content and structure in slides results in poorer quality of annotation compared to long, well-articulated texts. SPARQL is used to represent the question structure temporarily before verbalising, as also done in [3].

The first of the three varieties are gap-fill questions, generated by taking a sentence where a DBpedia resource is mentioned, replacing it with a blank and providing other similar DBpedia resources as distractors. The second variety of questions are generic MCQs, where distractors are generated using class hierarchies associated with a DBpedia resource. The third are *Jeopardy-style* questions, which include extra information. This information helps to generate more relevant distractors. Examples of the three varieties are shown below.

1. *The Eiffel Tower is located in ______.*
   a) Madrid b) Amsterdam ***c) Paris*** *d) Lyon*

2. *Lionel Messi is a:*
   a) Golf player b) Cricketer c) Gridiron football player ***d) Soccer player***

3. *I am a scientist. I starred in Atomic Power (film) and Julius Sumner Miller was influenced by me. Who am I?*

   a) Albert Einstein *b) Enrico Fermi c) J. Robert Oppenheimer d) Vannevar Bush*

[1]SlideWiki is an open source slide sharing platform http://slidewiki.org/
[2]DBpedia is a huge knowledge base extracted from Wikipedia http://wiki.dbpedia.org/

We discuss the related work performed in this field in Section 2, followed by the presentation of our approach in Section 3 and implementation in Section 4. In Section 5, we discuss the benefits and limitations of our approach along with future work that can be performed for potential improvements.

2 Related Work

Several researchers have used semantic web based approaches for question generation. For example, Papasalouros et al. [1] generate questions from domain ontologies. They use `rdf:subClassOf` to get the class hierarchies of OWL[3] classes. Al-Yahya [4] also uses domain ontologies to create three types of questions - *multiple choice*, *true/false* and *gap-fill*. The latter are generated by removing either the subject or the object of a triple. For the *true/false* type, false statements are created by replacing the subject or the object. *MCQs* are generated using triples in a similar sense. A combination of subjects and objects are chosen at random from a collection of triples to create distractors. Bühmann et al. [2] generate *Jeopardy-style*, *yes/no* and *multiple choice* questions. Their system, named *ASSESS*, performs automatic verbalisation of RDF graphs. Entity summarization provides key properties to identify resources with. This enables forming terse questions. Lastly, Concise Bound Descriptions[4] are used for fragmenting input knowledge to provide domain specific questions (domain chosen by user). Seyler et al. [3] generate *Jeopardy-style* natural language questions using knowledge graphs. The user chooses a topic and a difficulty level. A query is generated that would produce only a single result. It is then verbalised to create a question. They also generate multiple choice questions. The difficulty is calculated considering factors like popularity of the resource, selectivity of the triple pattern and coherence of the triples used.

The contribution of our work is an approach to generate MCQs (and distractors) for entities extracted from slides of any domain. Distractors are generated by either using data from DBpedia or finding matches from within the pool of entities. We extend the approach in [1] to generate more relevant distractors using deeper types and considering difficulty. Distractors generated for entities in [4] are random whereas we consider ones with similar types to the answer. Bühmann et al. [2] generate distractors using true and false triples. We also factor in the popularity of the answer to get distractors with similar popularity. For the third variety of questions, Seyler et al. [3] produce a question as well as distractors. Since the solution to our generated questions is not unique, we have more room to maneuver the distractors.

3 Question Generation

Now we discuss the structure of the generated questions, our strategy for generating them and the distractors for the answer.

All three varieties are generated based on a DBpedia resource identified by DBpedia Spotlight[5], in the content. Depending on which variety was chosen, a different algorithm is followed.

For the first variety, we take the sentence where the resource was found and replace it with a blank. The resulting statement constitutes the question text. For the second variety, the `rdf:type` of the resource is queried for. We use the `rdf:subClassOf` property of a selected

[3]Ontology Web Language `https://www.w3.org/OWL/`

[4]`https://www.w3.org/Submission/CBD/`

[5]A tool for annotating mentions of DBpedia resources in text `http://www.dbpedia-spotlight.org/`

`rdf:type` to get distractors. This is explained in more detail in Subsection 3.3. The *jeopardy style* question consists of an `rdf:type` of a resource and two triples that hold true for it.

Our approach uses a list of DBpedia resources returned by DBpedia Spotlight to generate questions. Spotlight is a good candidate since it provides the best F1 scores without over-annotating [5]. Out of the several `rdf:type`s that a resource has, one is selected as a reference to generate distractors. Let us call this the *base type*. The selection depends on the required level of difficulty. A shallow type will be used to create easy questions and a deep type for hard questions (explained in Subsection 3.2). Distractors are generated using different strategies for different varieties. For instance, if the answer is a resource, then distractors would be resources of the same `rdf:type` but having some properties that differ. In case the answer is an `rdf:type`, distractors are *sibling types*[6] that do not apply to the answer resource. Verbalisation of *jeopardy style* questions is performed using templates.

3.1 Semantic Annotation

Semantic annotation is not a part of our work but does affect the quality and correctness of the questions. The confidence parameter of DBpedia Spotlight is set to 0.6 for best results [5]. For *jeopardy style* questions, the type filter is used to annotate only mentions of type `DBpedia:Person`. The response from DBpedia Spotlight contains basic `owl:Class`es that can be directly used to generate distractors. A limitation of Spotlight is, that it is designed to work for long, formatted texts, which is not the case for slide content.

3.2 Type and Triple Selection

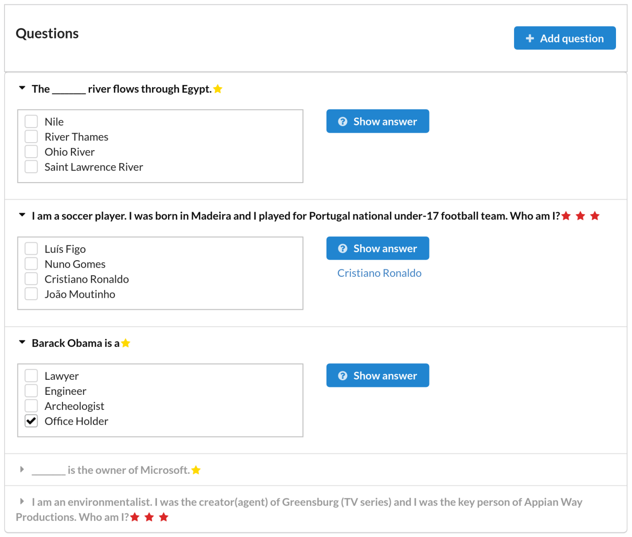
We select a type as the base type by calculating the count of the `rdfs:subClassOf*` results returned for each `rdf:type`. We call this count the *depth* of a type. There is a direct correlation between the depth of a type and how specific it is. For example, the type `DBpedia:SoccerPlayer` is an `rdfs:subClassOf` of `DBpedia:Athlete`, which is an `rdfs:subClassOf` of `DBpedia:Person`. Clearly, `SoccerPlayer` is more specific than `Athlete` since the former tells us more about the resource than the latter. This helps us control how much we want to delve into the subject (amount of difficulty). For example, generating a question about a soccer player is more generic than asking about a soccer player playing for *FC Barcelona*. Deeper types also lead to more closely related distractors. For example, *Lionel Messi* is a soccer player. He does not have as much in common with *Barack Obama* (both belong to type `Person`) as he does with *Cristiano Ronaldo*, who happens to be another soccer player and hence would prove to be a more fitting distractor in most cases. Based on this notion, we consider the most specific `owl:Class` as the base type of a resource for easy difficulty. In case even the most specific OWL type is too generic (e.g. `Person`), we get a random type between the depths of 11 and 14 (both exclusive). These depths usually contain YAGO[7] types that are similar in specificity to the most specific OWL types. For hard difficulty, we choose the most specific type a resource has as the base type. The depth is calculated by using the `rdf:subClassOf*` property and counting the number of types returned.

To form *jeopardy style* questions, we use two triples which hold true for the resource. The LinkSUM project query [6] is used to sort all triples in decreasing order of vrank value[8] of the subject if the resource is the object or vice-versa. For easy difficulty, two random triples are

[6]Types that are children of the same super-type, at the same level of depth

[7]A huge semantic knowledge base like DBpedia `http://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/`

[8]An RDF schema vocabulary for ranking computations `http://lov.okfn.org/dataset/lov/vocabs/vrank`


Questions

+ Add question

The ______ river flows through Egypt.★

- ☐ Nile
- ☐ River Thames
- ☐ Ohio River
- ☐ Saint Lawrence River

Show answer

I am a soccer player. I was born in Madeira and I played for Portugal national under-17 football team. Who am I?★ ★ ★

- ☐ Luís Figo
- ☐ Nuno Gomes
- ☐ Cristiano Ronaldo
- ☐ João Moutinho

Show answer

Cristiano Ronaldo

Barack Obama is a★

- ☐ Lawyer
- ☐ Engineer
- ☐ Archeologist
- ☑ Office Holder

Show answer

______ is the owner of Microsoft.★

I am an environmentalist. I was the creator(agent) of Greensburg (TV series) and I was the key person of Appian Way Productions. Who am I?★ ★ ★

Figure 1: User interface of the questions section in SlideWiki

chosen from the top 10 triples. Having a relatively large vrank value, these triples describe facts about the resource that it is well-known for. In case of hard difficulty, a different approach is used. An initial vrank range of lower bound 0.2 and upper bound 0.5 is chosen. A triple is searched for within this vrank range. If found, it is selected. Else, the lower bound is shifted to the upper bound value and the upper bound value is doubled. This is repeated until a triple is found or the upper bound value crosses the maximum vrank value. In either difficulty, it is ensured that both triples are not duplicates and neither are their properties. Hence, we make sure that two different aspects about the resource are asked for in the question.

3.3 Distractor Generation

Distractors are divided into two categories: in-text and external.

In-text distractors are used for the first and third varieties where the answer and distractors are resources and not types. They are other resources also found in the text which have the same type as the concerned resource. A `MultiMap` is used to hold resources grouped by their types. Hence, each resource is grouped with its in-text distractors.

External distractors are generated using the base type. For the first variety of questions, the distractors are just other resources of the base type whereas for the second variety, they are sibling types[6] of the base type. For *jeopardy style* questions, just like the first variety, other resources of the base type are found but the two triples, that the question is made of, are also considered. For easy difficulty distractors, both should hold false and for hard, either one should hold true.

For all cases, the results with the closest popularity to the answer are chosen. To do so, we calculate the difference d between the DBpedia Pagerank[9] value of the answer and each result. We sort the results in increasing order of d and choose the top three. A check is always made to ensure that a resource has a Pagerank or not. In case the Pagerank is not available for the answer resource, the top three resources based on Pagerank are used.

[9] `http://people.aifb.kit.edu/ath/`

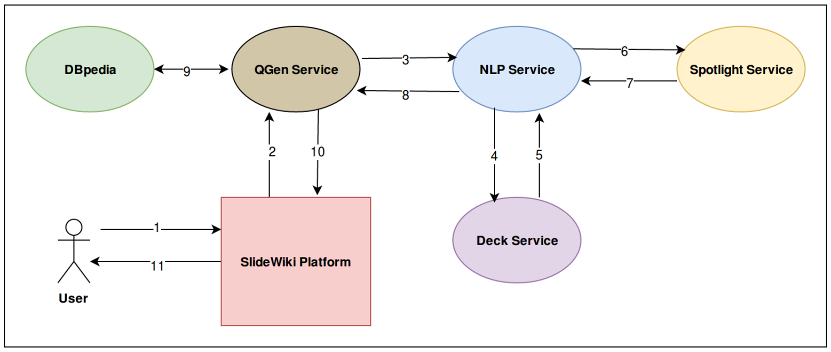


Figure 2: High level architecture and work flow of our system with SlideWiki

3.4 Verbalisation

Verbalisation is performed for the *jeopardy style* questions only. Manually created templates for ten to fifteen commonly occurring predicates are used. These templates can be applied irrespective of the position of the answer resource in the triple. In case no template is found for a predicate, some patterns like the suffix *-or* or keywords like *by* are searched for. Commonly occuring predicates like *known for* or *influenced* are verbalised, considering the position of the answer in the triple. For example, if the answer is the subject and the predicate ends with the pattern *-or* (e.g. *Creator*), we verbalise it as *"I am the creator of ..."*. In case none of the strategies work, a generic verbalisation *"I was the **predicate** of **subject/object**"* is applied. Verbalisation can be easily extended to include more templates as well as patterns.

4 Implementation

Now we discuss some implementation details about the system. The system is written in Java 8. We use Apache Jena[10] to communicate with SPARQL endpoints. Maven is used as the build tool. The system is hosted on a Tomcat 7 docker container. Docker is used since the project will be deployed as a microservice as part of the SlideWiki backend architecture. A microservice architecture ensures autonomous parts i.e. each microservice can be changed internally or new ones can be added without affecting the rest of the system.

Our system is designed to communicate with another microservice called the NLP microservice. This microservice communicates with the DBpedia Spotlight service which could either be their web service[11] or a local instance (also running as a microservice). The NLP service provides clean text extracted from slides and spotlight results.

To communicate with our system, a REST api is provided with the URLs comprising of path parameters like question type and difficulty level. A SlideWiki deck id can be sent to the service to generate questions for the entire deck. Plain text can also be sent in case the user selects some text and requests for questions for this text.

Figure 2 shows how our system communicates with the other microservices and the SlideWiki platform. Here are the steps of the work flow:

[10]Apache Jena is open source Java framework for building Semantic Web and Linked Data applications `https://jena.apache.org/index.html`

[11]`http://www.dbpedia-spotlight.org/spotlight-api`

1. User requests questions for a particular deck
2. The SlideWiki platform makes a call via our REST api
3. Our system asks the NLP service for annotation results and text of the concerned deck
4. The NLP service asks for deck contents from the deck service
5. The deck service returns deck contents
6. The contents are sent to DBpedia spotlight for annotation
7. DBpedia Spotlight returns the results
8. Annotation results along with other NLP results are returned
9. The questions are generated by querying DBpedia
10. Questions are returned as JSON
11. The platform presents the questions to the user as shown in Figure 1

5 Discussion and Future Work

We now list some observations, limitations and their potential solutions. A high number of slides in a set leads to better annotation and hence, better distractors. The second and third variety prove to be a helpful addition since they don't use text from the slides and hence also work well for slides with short phrases or one word bullet points. Incorrect types for a resource (second variety) reveal discrepancies in knowledge base data. Looking for other resources of a shallow type (for easy difficulty) can take time. Shallow types, being generic, are present in a large number of resources. Since our algorithm compares popularity, a larger set requires more time. Distractors for the first variety of questions are generated simply by looking for similar resources irrespective of the content of the question. This could lead to distractors being correct answers for the question in some cases. A solution could be to use a natural language to RDF converter. This would provide the context of the sentence and help generate accurate distractors. While selecting a base type for easy difficulty, if the most specific OWL class is too generic, some times a type between a given depth range is randomly selected. The type may or may not identify the resource well and hence may produce distractors that are not very accurate. A corpus could be used to mine data and learn about salient types [3]. Some triples included in a *jeopardy style* question belong to only the given resource for the given base type. Hence, distractors are generated only considering the base type and not the triples. An alternative approach can be used by also looking for resources that have the triples as true but are not of the base type. e.g. For base type scientist, who has acted in a film, the distractors could also be regular actors (not scientists) who acted in the same film. Our approach can be easily extended to generate many more question varieties like in [1].

References

[1] Andreas Papasalouros, Konstantinos Kanaris, and Konstantinos Kotis. Automatic generation of multiple choice questions from domain ontologies. In *e-Learning*, pages 427–434. Citeseer, 2008.

[2] Lorenz Bühmann, Ricardo Usbeck, and Axel-Cyrille Ngonga Ngomo. Assessautomatic self-assessment using linked data. In *International Semantic Web Conference*, pages 76–89. Springer, 2015.

[3] Dominic Seyler, Mohamed Yahya, and Klaus Berberich. Knowledge questions from knowledge graphs. *arXiv preprint arXiv:1610.09935*, 2016.

[4] Maha Al-Yahya. Ontoque: a question generation engine for educational assesment based on domain ontologies. In *Advanced Learning Technologies (ICALT), 2011 11th IEEE International Conference on*, pages 393–395. IEEE, 2011.

[5] Pablo N Mendes, Max Jakob, Andrés García-Silva, and Christian Bizer. Dbpedia spotlight: shedding light on the web of documents. In *Proceedings of the 7th international conference on semantic systems*, pages 1–8. ACM, 2011.

[6] Andreas Thalhammer, Nelia Lasierra, and Achim Rettinger. Linksum: using link analysis to summarize entity data. In *International Conference on Web Engineering*, pages 244–261. Springer, 2016.