# Knowledge Graphs

Wikinator - A Knowledge Graph Project

presented by
Vitor Faria, Yannick Laudenbach
Matriculation Nr.: 1844490, 1653030

submitted to the
Data and Web Science Group
Prof. Dr. Heiko Paulheim
University of Mannheim

October 2022

# Contents

# 1 Introduction

The Akinator is a video game developed by French company Elokence. It is a virtual genie, attempting to determine what fictional or real-life character, object, or animal the player is thinking of by asking a series of Yes/No questions. The Akinator is currently available in 16 languages, and it has already been played over 500 million times in its English version, according to their own landing page [1].

The Akinator's main challenges during the game are choosing the best possible next question based on accumulated knowledge, and making a proper guess after a few questions. How these tasks are actually done is still unknown out of its company, but both tasks could benefit from vast Knowledge Graphs (KG) with many entities and properties, such as open Knowledge Graphs built from Wikipedia data. Some examples of public Wikipedia-based KGs are DBPedia [2], Yago [4] and Wikidata [5]. These well known Knowledge Graphs are often considered similar in nature and coverage, although they have quite a few differences in terms of number of classes, number of instances per class and number of properties per instance [3].

To prove that such sources are enough to provide a similar game, this project introduces the *Wikinator* - an Akinator variant based solely on DBPedia data. Its source code is publicly available on GitHub.

## 2 Datasets Used

The used dataset for this application is DBpedia. DBpedia was selected because it contains numerous entries for persons and fictional characters. Its ontology also works with subclasses, which allow for an effective filtering of entities. In addition, its entries consist of many references and links to other pages, which is the base of the guessing and evaluation process of the Wikinator, which is explained in more detail in section 4.3

Although Wikidata is the most suitable KG for person data [3], DBPedia has also a great advantage of reproducing Wikipedia resource names as their URLs. This is helpful for the task of translating the results of a SPARQL query into a question in natural language. Yago and Wikidata, on the other hand, uses numerical IDs to reference its resources, which would require some extra effort in this translation.

Each question answered by the players can then be translated to a SPARQL query against DBPedia's endpoint, by adding extra filters to the WHERE clause of a template query. In this sense, the application does not need to store all information about Person and Fictional Character entities in a database, just to query the original source and retrieve aggregated results.

## 3 Application Domain and Goals

Similarly to the original Akinator, the application is a single-player game and targets any public on the internet, and should entertain its players by asking a sequence of smart questions, but as few as possible to make a proper guess. The measure of success for such task are accuracy and the average number of questions until guess. It should be able to guess correctly a list of selected people and fictional characters within approximately 20 Yes/No questions, which is comparable to the original Akinator's performance. By succeeding, the Wikinator can demystify the black box Artificial Intelligence behind Akinator, and prove that Knowledge Graphs are suitable for intelligent games even with simple SPARQL queries.

# 4 Techniques Used

To narrow down possibilities, rule out persons and guess the correct person, the Wikinator has to ask the correct questions first. It therefore uses three different techniques to limit the amount of possible answers, which are presented in the following sections.

## 4.1 Class Hierarchy Analysis

An intuitive and reasonable starting point when having little to no information about the entity is to look for the most specific class this entity belongs to, searching on DBPedia's class hierarchy. It uses the subclasses of the main type the user has chosen at the start of the round. The algorithm reads all subclasses of the main type and asks whether the user's person is of this type. The algorithm starts asking about the subclass with the most entries, because it should have the higher probability. If the entity belongs to this subclass, the algorithm goes deeper into the hierarchy relation and asks about its subclasses. If not, the Wikinator asks about the next subclass with the highest number of instances. This method continues in a loop until there are no further subclasses to ask for. If that is the case, it starts asking a different type of questions: about their predicates-object relations.

## 4.2 Analyzing Predicate-Object Relations

After all subclasses have been analyzed, a different algorithm runs, searching for all distinct predicate-object relations among the remaining candidates. A query selects all predicate-object relations and counts the occurrences of the exact relations. The following code shows an exemplary SPARQL query for this operation, given the information that the person is of type *Person* but not of type *SoccerPlayer*.

```
SELECT
    ?predicate
    ?object
    (COUNT(?candidate) AS ?occurrences)
WHERE {
    ?candidate ?predicate ?object.
    ?candidate a dbo:Person.
    FILTER NOT EXISTS {?candidate a dbo:SoccerPlayer}
}
GROUP BY ?predicate ?object
ORDER BY DESC (?occurrences)
```

| **Predicate** | **Object** | **Appearances** |
|---|---|---|
| http://purl.org/linguistics/gold/hypernym | http://dbpedia.org/resource/Cricketer | 13384 |
| http://dbpedia.org/ontology/battingSide | Right-handed | 13310 |
| http://dbpedia.org/property/batting | Right-handed | 13018 |
| *http://dbpedia.org/property/columns* | *1* | *9746* |
| http://dbpedia.org/property/tenfor | 0 | 9225 |
| http://dbpedia.org/property/column | http://dbpedia.org/resource/List_A_cricket | 7242 |

Table 1: Result the algorithm chooses the next question from. The next question chosen is marked cursive. This question is chosen, as it is the closest to the ideal occurrence of an predicate-object relation.

The ideal predicate-object relation to be asked for the next question is the one that could split the candidates into two groups of equal size. Therefore, the ideal number of occurrences is equal to the total number of instances divided by two. After calculating the ideal number of occurrences, the result of the query is ordered ascending by the distance to that ideal occurrence. The first relation will be used in the next question for the user. In case the user does not know the answer to this question, the next closest relation is used as next question. If the answer is either ‘Yes‘ or ‘No‘, this new pattern is added to the WHERE clause of the next SPARQL query, iteratively. This algorithm runs, until there are no further questions to be asked, or in case the user decides to let the Wikinator guess the entity based on the information gained so far. Table 1 shows a possible result the algorithm could return. Figure 1 shows the question the user would get from this query.

### 4.3 Making a Guess

To make the guess, a new SPARQL query is fired with all the information collected so far in its WHERE clause. However, instead of counting predicate-object relations, this query counts the number of ”wikiPageWikiLinks”, which correspond to the number of internal hyperlinks in Wikipedia pages. This method relies on the following assumption:

*A person that has more in-going ”wikiPageWikiLinks” is more famous in the real-world, which leads to that person being chosen more frequently by users.*

Based on this assumption, the person with the higher number of in-going links should be the

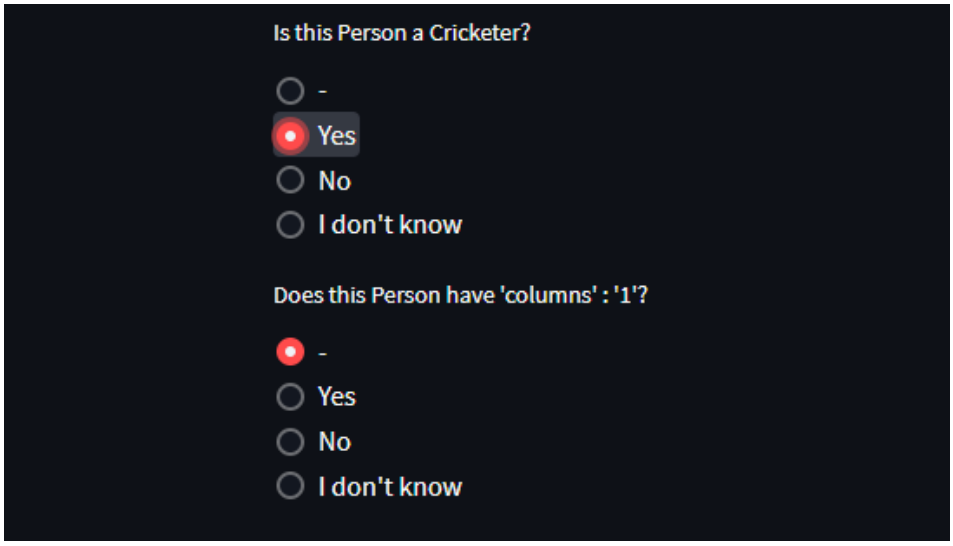


Figure 1: Example question the user is asked based on the information the algorithm got from querying for predicate and object

first guess, as the probability for this person being chosen is the highest. An example guessing-query is structured as follows:

```
SELECT
    ?candidate
    (COUNT(?link) AS ?links)
WHERE {
    ?link dbo:wikiPageWikiLink ?candidate .
    ?candidate a dbo:Person .
    ?candidate a dbo:Athlete .
    FILTER NOT EXISTS {?candidate a dbo:SoccerPlayer}
}
GROUP BY ?candidate
ORDER BY DESC (?links)
LIMIT 5
```

This query returns results as described in Table 2, with the top 5 athletes in terms of ingoing Wikilinks, except for Soccer Players. It can be noticed by these results that all athletes with a high number of links are indeed famous in their respective Sports.

| Candidate | Links |
|---|---|
| http://dbpedia.org/resource/Roger_Federer | 2121 |
| http://dbpedia.org/resource/Serena_Williams | 2075 |
| http://dbpedia.org/resource/Rafael_Nadal | 1884 |
| http://dbpedia.org/resource/Venus_Williams | 1779 |
| http://dbpedia.org/resource/Novak_Djokovic | 1773 |

Table 2: Results for the example SPARQL query to make a guess, with the top 5 Athletes but non-Soccer Players with the most ingoing wikilinks.

## 5 Example Results

Table 3 and 4 show, how well the Wikinator can guess the correct answer and how many questions are needed for it. The persons have been guessed based on their in-going ”wikiPageWikiLinks” and range from the most known to the least known person, in steps of 25% popularity of the most known person. The results for real-world persons are all of type *Athlete*, as only querying for type *Person* would return countries, as countries are of type *Person* in DBpedia.

The results in the table 3 and 4 show, that the amount of questions asked is not solely based on the popularity, but rather on the amount of types the person has, explaining, why ”Alistair Dewhurst” has been guesses quicker than ”Fernando Verdasco” for the real-world-persons.

| Person | in-going WikiPageWikiLinks | Questions asked |
|---|---|---|
| Roger Federer | 2121 | 1 |
| Fernando Verdasco | 1560 | 25 |
| Jarkko Nieminen | 1059 | 24 |
| Vince Lombardi | 530 | 12 |
| Alistair Dewhurst | 1 | 18 |

Table 3: Results for real world persons with different popularity based on the ”WikiPageWikiLink” assumption.

| Character | in-going WikiPageWikiLinks | Questions asked |
|---|---|---|
| Batman | 3723 | 0 |
| Spider-Man | 2748 | 10 |
| Wonder Woman | 1695 | 5 |
| Darth Vader | 929 | 15 |
| Mamur Zept | 1 | 45 |

Table 4: Results for fictional characters with different popularity based on the ”WikiPageWikiLink” assumption.

# 6 Known Limitations

Running Wikinator up on DBpedia's graph comes with several limitations. Although DBpedia has a considerably large database, a lot of the data is often incomplete, duplicated or even inconsistent, leading to contradictions. This happens not only due to the Open World and the Non-Unique Naming Assumptions, but also due to not up to date or even lack of information in the main source Wikipedia. For instance, there are instances of classes that should be disjoint and entities in wrong classes.

Furthermore, this application only allows to query for real-world persons or fictional characters. The following section will explain these and other challenges, the solution to it and the limitation the solution might entail.

## 6.1 Inconsistent Data

The data provided by DBpedia is often inconsistent within different people with the same type. For example, person *A* is of type *Person, Athlete* and *Soccer Player*. Another person *B*, however, is only of type *Soccer Player*, but not of type *Athlete*. As *Soccer Player* is a sub-class of *Athlete*, one would expect a *Soccer Player* to always be an *Athlete* as well. A Soccer Player who is neither an *Athlete* nor a *Person*, can thus not be queried for in the database, as the starting question requires someone to be of type *Person* or *Fictional Character*

This problem also holds true the other way around. Sometimes a country, like ”USA” or ”Germany” is of type *Person* and are thus in the result set of the first question.

This problem cannot be fully solved, however, by analyzing subclasses and their subclasses partly independently of each other, people who are *Soccer Players* but no *Athlete* can still be found with the help of their type. A person who is not of type *Person* cannot be found with the help of the Wikinator.

## 6.2 Duplicate, but Varying Data

In the DBpedia database, some persons have more than just a single entry. In theory, as long as both entries have the exact same data, this would not cause an issue for the guessing of the Wikinator. However, two entries having different data causes difficulties for the user. For example, DBpedia holds two entries for Lewis Hamilton: The first entry is about Lewis Hamilton as a person himself. The second entry is about his Formula 1 racing career. Nonetheless, his career entry holds more valuable information about him, than his actual entry. This includes the type *Racing Driver* or also *Athlete*. This makes it difficult for the algorithm to find Lewis Hamilton, as a user would tend to choose the entry about Lewis Hamilton as a person.
In theory, the Wikinator can still guess the correct person, given it has enough varying data to other entities. If this is not the case, then the person will not be guessed correctly.

## 6.3 Referenced Data

DBpedia contains many references to other databases, such as the Yago database. This increases the amount of data available for querying, however this data is not readable, until the reference has been found in Yago. Figure 2 shows such a reference. Although it clearly states, that the reference is made to Yago, it cannot be found within Yago, as it leads to a page not existing in Yago itself. Asking the user for the types seen in figure 2 are difficult to interpret for the user. To resolve this issue, the Wikinator filters out all Yago references, in addition to all wiki links to

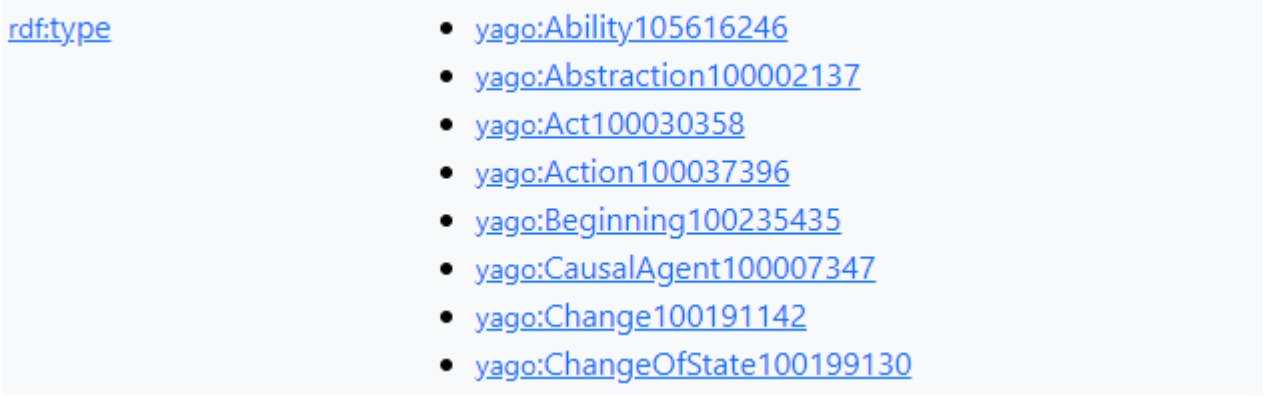


Figure 2: Example of data that is referenced to Yago

Source: https://dbpedia.org/page/Spider-Man

pages, before evaluating its next question.

### 6.4 Limited Categories

Currently, the user can choose between a real-world person or fictional characters. However, as the code for the algorithm is completely dynamic and not hard-coded, other categories, such as buildings or events could be added, given the data is maintained in a similar manner.

# 7 Lessons Learned

Public Knowledge Graphs are a very rich source of information that can be easily retrieved using SPARQL endpoints, and proved themselves to have enough knowledge for a quizz game such as the Akinator. Both of its main tasks could be translated to simple SPARQL with some kind of aggregation, to be used in an iterative Q&A.

Some challenges could be successfully overcome, such as translating SPARQL results into natural language questions, and Yes/No answers into new patterns for the next query. Using the number of wikilinks to rank the candidates and the ideal number of predicate-object relations were also adequate solutions for the challenging iteration process. However, some extra filters had to be applied on the query results a posteriori, in order not to make meaningless questions.

Although the procedure is functional and reasonable, it still struggles when it comes to inconsistencies in DBPedia's classes, such as people that are not of type 'Person', and entities that are not people but do have type 'Person'. It also does not benefit so much from information that are intuitively important for the guessing process and are quite obvious to humans, but barely exist in DBPedia's triples, such as gender. Similarly, when users answer 'Yes' to a question, and the specific entity they are thinking of does have that property indeed, but not in DBPedia, it will never guess correctly.

To deal with all these known limitations, it would be necessary to combine DBPedia with other KGs, to have more information available about each instance. Also, the users' answers should not always be considered as ground truth, but most likely truth. This is already implemented in the original Akinator, and could fit better the Open World Assumption behind KGs.

# 8 Small Outlook

To improve the user experience of the Wikinator, multiple features could be implemented in the future. Creating a lexicon, which translates the most common predicates and objects into natural language could greatly increase the usability of the application. However, not every edge-case or rare scenario can be covered in the lexicon, requiring some other treatments for special cases. Furthermore, a list with all the answers to the questions could be shown at the side, showing the user how many, which questions and how the questions have been answered at one glance.

Another useful feature is that one could implement further answering possibilities, including options like ”Probably” or ”Probably Not”. A newly created algorithm could therefore rank questions based on their answers and how sure the user was, when answering them. This could eliminate the assumption that every answer is ground truth, as mentioned in 7.

As a last feature, is the automated guessing. Currently, the user decides when the Wikinator should take the guess, however, if only one person is left in the result, the Wikinator should automatically guess. Additionally, the more questions have been answered the more likely the Wikinator could be to take a guess, even if multiple results are possible. A ranking for the difficulty of the guess could also be implemented, letting the user know if the person chosen was difficult, or rather easy, making the application even more interactive.

## References

[1] Elokence: Akinator. https://en.akinator.com/, accessed: 2022.12.08

[2] Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia – A Largescale, Multilingual Knowledge Base Extracted from Wikipedia. Semantic Web Journal (2013)

[3] Ringler, D., Paulheim, H.: One Knowledge Graph to Rule them All? Analyzing the Differences between DBpedia, YAGO, Wikidata and co. (2017)

[4] Suchanek, F.M., Kasneci, G., Weikum, G.: YAGO: A Core of Semantic Knowledge Unifying WordNet and Wikipedia. 16th international conference on World Wide Web (2007)

[5] Vrandečić, D., Krötzsch, M.: Wikidata: a Free Collaborative Knowledge Base. Communications of the ACM (2014)

Mannheim, December 11th 2022.