Inside Quasimodo: Exploring Construction and Usage of Commonsense Knowledge

Julien Romero, Simon Razniewski

Inside Quasimodo: Exploring Construction and Usage of Commonsense Knowledge

Julien Romero
julien.romero@telecom-paris.fr
LTCI, Télécom Paris, Institut Polytechnique de Paris
Palaiseau, France

Simon Razniewski
srazniew@mpi-inf.mpg.de
Max Planck Institute for Informatics
Saarbrücken, Germany

ABSTRACT

Quasimodo [10] is an open-source commonsense knowledge base that significantly advanced the state of salient commonsense knowledge base construction. It introduced a pipeline that gathers, normalizes, validates and scores statements coming from query log and question answering forums. In this demonstration, we present a companion web portal which allows (i) to explore the data, (ii) to run and analyze the extraction pipeline live, and (iii) inspect the usage of Quasimodo's knowledge in several downstream use cases. The web portal is available at https://quasimodo.r2.enst.fr.

KEYWORDS

datasets; knowledge base; visualisation; commonsense

1 INTRODUCTION

Commonsense knowledge (CSK) is a recurring theme of AI introduced in 1960 by McCarthy [7]. Researchers such as Doug Lenat focused their entire career on this area [4, 5]. More recently, deep learning models and their limits raised the question of explainability and the urge to develop new systems with knowledge at their heart. Marcus [6] advocates the development of a hybrid, knowledge-driven, cognitive-model-based approach to create robust artificial intelligence. He claims that crisp knowledge, and in particular commonsense knowledge will have a fundamental role to play.

Recently, we [10] introduced the commonsense knowledge base *Quasimodo*. The system leverage human curiosity expressed in query logs and question answering forums to extract commonsense, and crowdsourcing experiments showed that it significantly outperformed existing resources like ConceptNet in terms of coverage. In this paper, we introduce a companion web portal which enables a comprehensive exploration of Quasimodo's content and its construction. In particular, our contributions are:

(1) Development of a scalable architecture for knowledge base visualisation;
(2) Visualisation of the extraction pipeline of Quasimodo;
(3) Implementation of several applications on the Quasimodo data.

First, we review in Section 2 the state of the art of the methods used to display knowledge graphs for the main public knowledge bases. Then, we briefly recall what Quasimodo is and how it works in Section 3. We introduce the companion web portal of Quasimodo in Section 4, and present the demonstration experience in Section 5. This web portal is composed of several parts: an explorer, a visualisation for the extraction pipeline, a SPARQL endpoint and applications such as question answering or games like *Play Taboo!* and *Codenames*.

2 PREVIOUS WORK

We present here some existing system for knowledge base visualisation. Most of them allow exploring the raw content and provide a SPARQL endpoint—however, very few focus on applications.

The most typical way to display a knowledge base is to print it as a table composed of three columns: Subject, predicate, object. ConceptNet [12], WebChild [13], TupleKB [3], Atomic [11], Comet [1] and Quasimodo [2] provide a CSV file in that style. Often, it is convenient to group the statements by subject on a separated page, and thus to omit the first column. We also regularly observe that systems tend to group the statements by predicates.

There exist more exotic ways to display a knowledge base. Yago [9] chooses to give a glimpse to the relations attached to a given subject through a star-shaped graph. Their companion web portal displays an SVG of this graph. The graph structure is very natural when we deal with knowledge bases. Some third-party websites such as Geneawiki[1] or the Wikidata Graph Builder[2] use this representation to display relations between entities. However, due to the size of the graphs, it becomes tough for a human to find relevant information.

In our system, we choose to use a table representation where we added additional columns such as the polarity (positive or negative) of a statement, an attached modality (e.g. sometimes, always, never) or a score.

Many systems also provide a simple search interface. More interestingly, some give access to a SPARQL endpoint to write complex queries. Finally, the websites offer an easy way to download data in different formats. We provide all these functionalities.

Third-party web portals generally provide the applications associated with a knowledge base. For example, Inventaire[3] uses Wikidata to extract information about books and The Art Browser[4] uses Wikidata to display art information. Wikidata groups various projects related to their website[5], and in particular about visualisation. In this paper, we include several applications near the data visualisation, which can make the user more familiar with the knowledge base.

[1] https://tools.wmflabs.org/magnus-toolserver/ts2/geneawiki/
[2] https://angryloki.github.io/wikidata-graph-builder/
[3] https://inventaire.io
[4] https://openartbrowser.org
[5] https://www.wikidata.org/wiki/Wikidata:Tools

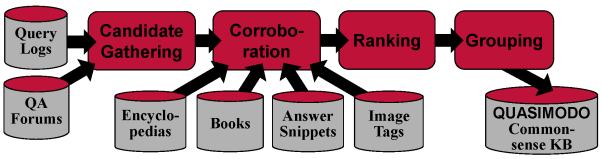

Figure 1: Quasimodo extraction pipeline.

3 QUASIMODO

Quasimodo is a commonsense knowledge base generated from query logs and question answering forum. It uses a pipelined architecture.

The extraction pipeline is detailed in the original paper [10] and is summarized in Figure 1. We recall here the main steps.

Candidate Generation. We extracted from query logs (through the autocompletion of Bing and Google) and question answering forums (Answers.com, Reddit, Quora) why and how questions. Then, we turned them into statements and used OpenIE techniques to extract triples. Next, we normalise these triples by using several methods such as the lemmatisation of the subject and the predicate, the removal of noisy words or the elimination of personal terms.

Corroboration We compared the statements to external sources to confirm (or not) the correctness. For example, we checked how often the predicate and the object appear on the Wikipedia page of a given subject. We also used image tags and captions to find links between a subject and an object.

Ranking We combined the signals from all sources using a supervised algorithm to obtain a score for each fact. Then, we generated additional metrics that take into account the structure of the knowledge base.

Grouping The statements are bi-clustered by subject-object and predicate to normalise them. This step is explained in more details in [8] and so will not be part of this demo.

Resulting Data As of May 2020, Quasimodo is composed of approximately four million statements for over 95.000 different subjects, making it more than ten times bigger than ConceptNet. In [10], it was shown that its precision is significantly higher than Webchild, and that its recall significantly exceeds all other commonsense knowledge bases.

4 QUASIMODO WEB PORTAL ARCHITECTURE

We packed our demo as a web portal, accessible at https://quasimodo.r2.enst.fr. We use Nginx to manage connections, obtain HTTPS accesses and perform reverse proxy to the internal components.

To make our system scalable and reusable, we decomposed it into Docker containers. Docker containers are light and independent packages that contain everything to run an application. A developer can use them as building blocks for more complex applications deployed on a single or several computers. Our application runs on a single machine, and so a docker-compose file is used to link the entire system. Our system can be deployed on several computers as it is compatible with solutions such as Kubernetes that automatically scale each container according to its needs.

Let us enumerate the different containers present in the application. First, we wrote the core of the web portal (we call it the back-end) in Python using Flask. Flask is a lightweight micro web framework which comes with numerous additional packages. The back-end module orchestrates all the actions. In particular, it is linked to a container encapsulating a PostgreSQL database which stores Quasimodo. It is also linked to a container running a Redis database and which is used as a Job Queue (see Section 5.2). An arbitrary number of asynchronous workers on separated containers can connect to this job queue and execute tasks. Finally, we also have a particular container hosting a SPARQL endpoint: Oxigraph.

A user accesses our companion web portal through a front end which uses Bootstrap4 and simple HTML, CSS and Javascript. The SPARQL endpoint is also accessible independently of the web portal. We summarise the general architecture of the web portal in Fig. 2.

We tested most of the components of our system using the Pytest library and Selenium. Selenium is a framework to emulate a browser such as Chrome or Firefox. The code is freely accessible on Github[6] where all the containers and the docker-compose file are also available. Finally, we used Jenkins to run a pipeline of tests ensuring the validity of each component. This pipeline gets executed every time the git repository receives a push. A Docker container encapsulates the pipeline which runs the tests.

The web portal, the asynchronous workers and the SPARQL endpoint run on a single virtual machine which has access to 8 Virtual CPU of 2.6GHz and 16GB of RAM.

5 DEMONSTRATION EXPERIENCE

5.1 Exploring and Searching Commonsense Knowledge

The data is stored using a relational database (PostgreSQL) which provides a fast way to retrieve information. We use a single table to store all statements, with columns for subject, predicate, object, modality, polarity, example sentences and metrics (scores).

We provide a simple visualisation for the statements in Quasimodo as a table containing columns for the subject, predicate, object, modality, polarity (is it a positive or a negative statement) and scores. In difference to KBs with fixed predicates, like Wikidata or ConceptNet, we organize the open predicate space by sorting statements by scores. Besides, we added the possibility to give positive or negative feedback about a statement, which could be used to refine supervised models. We also implemented a page per statement to display all information about it, and in particular the sentences that generated the statement and which sources they were derived from.

To traverse its content efficiently, a search function is available, which allows filtering the statements by subject, predicate, object and polarity. The search returns the number of matching statements and a table displaying them as explained above. In Figure 3, we show the top statements for the subject "elephant".

5.2 Extraction Pipeline Visualization

Quasimodo introduces an extraction pipeline to extract and process commonsense knowledge from various sources. In Section 3, we briefly presented this extraction pipeline. Each module in this workflow is itself composed of several sub-modules for performing specialized tasks. The reader can find a list of these components

[6] https://github.com/Aunsiels/demo_quasimodo

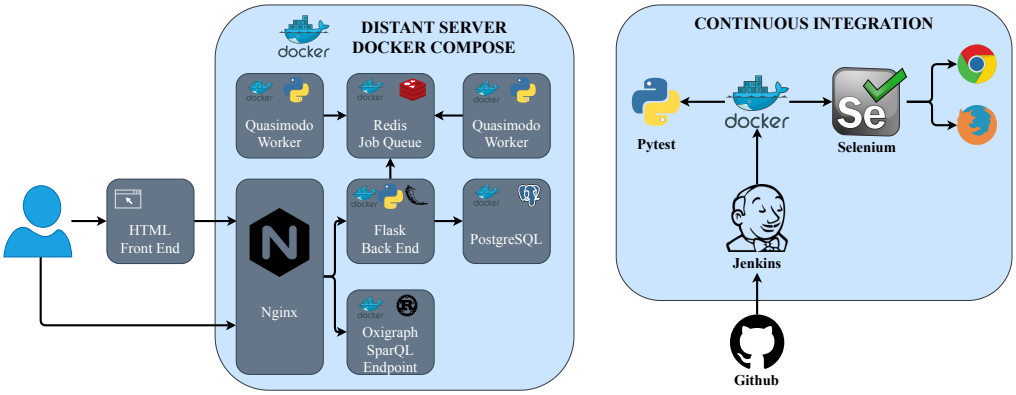

Figure 2: The web portal architecture.

Subject	Predicate	Object
elephant	live in	africa
elephant	reject	baby
elephant	have	large ears
elephant	have	good memory
elephant	has_body_part	hair
elephant	be in	africa

Figure 3: Top Quasimodo statements for elephants.

Assertion Generation
Google Autocomplete Duration: 2 minutes, 20 seconds
Bing Autocomplete Duration: 1 minutes, 12 seconds
Yahoo Questions Duration: 1 minutes, 6 seconds
Answers.com Questions Duration: 1 minutes, 35 seconds
Quora Questions Duration: 1 minutes, 4 seconds
Reddit Questions Duration: 0 minutes, 40 seconds

Subject	Predicate	Object
elephants	have	trunks
elephants	have	four feet
elephants	do well	in school
elephants	have	big ears

Figure 4: Top-level view of the extraction pipeline visualization.

in the original paper [10] and in the code provided with it.[7] This extraction pipeline, however, is very dense, and it can be not easy to understand the effect of each part. The present web portal therefore gives further insights into the entire process. In particular, we offer the possibility to run the extraction pipeline for a given subject. As most extraction sources must answer in a limited time, we launch the extraction pipeline in an asynchronous task using Redis Queues. Redis is an in-memory NoSQL database storing key-value couples. When we want to get the extraction pipeline information for a subject, we push a new job on a queue. Then, idle workers specialized in extraction pipeline execution come and read the pending task and start the extraction pipeline. Once they are done, they write back the details of the execution in the queue. They record these details every time a module or sub-module is executed, providing insights even if the extraction pipeline is still running.

The workers are based on the code given with Quasimodo and are encapsulated inside Docker containers. So, they can easily be duplicated, even on separated machines.

The back-end of the web portal has access to the status of the jobs and the currently available information. We display the details of the extraction pipeline on a web page showing the different stages of the extraction pipeline and the statements which are generated, modified or deleted. Besides, we also print the time spent in each module and sub-modules. The information of the extraction pipeline can be very dense, even for a single subject. So, the user has to choose in the interface a particular sub-module to display. We display the results as a table for the statements which were created, modified or deleted by the considered step. Figure 4 shows the beginning of the extraction pipeline of the subject *elephant*.

We intentionally omitted the scoring phase as it must access all generated statements, for all subjects. Executing the extraction pipeline for a given subject takes approximately 30 minutes on our machine.

5.3 SPARQL Endpoint

We offer a SPARQL UI and endpoint https://quasimodo.r2.enst.fr/sparql. The UI is available on the web portal, and the endpoint is callable by any program. Nginx orientates the query to detect whether we are accessing the SPARQL interface or the main web portal. Indeed, we put these two components on two separated containers. We dockerised Oxigraph[8], a SPARQL endpoint written in Rust which is also used by Yago. It has the advantage to be very easy to use and very fast. We transformed our data into N-triples,

[7] https://github.com/Aunsiels/CSK

[8] https://github.com/Tpt/oxigraph

at the cost of losing information such as the modality, the polarity and the scores.

5.4 Play Taboo!

Taboo is a game in which a player must make other players guess a word without using a list of forbidden words. In this demo, we provide an interface to play Taboo with Quasimodo. When a user starts a new game, the web portal sends him a card. Then they must use a chat interface to give clue words to Quasimodo. Every time the user presses the *Make a Guess* button, the system tries to a find a relevant word.

The algorithm used in the back end is simple. First, the database is filtered using the words given by the user. Then, we group the results by subjects, and we aggregate the scores using a sum or a max function, for example. We finally return the best subject, under the condition that we never tried it before.

In addition to this game, we also provide the functionality to generate Taboo cards for any subject. We perform this generation by taking the most relevant objects associated with a subject by combining the scores.

5.5 Codenames

Codenames is a game designed by Vlaada Chvátil. It opposes two teams that must find their special agent before the other team. The agents are hidden behind codenames, which are simple words. Each team has a spymaster which must give a clue to the rest of the team (the operatives) to help us reveal the spies. For example, a spymaster can say *blue, 2* to help its companion guess the words *sky* and *sea*. The choice of the word must be made very carefully not to discover the agents of the other team.

In 2019, the Foundation of Digital Games conference hosted a competition: *The Codenames AI competition*[9]. However, we were not able to find the results of this competition. The presenters suggested that people should use word embeddings. This solution can be powerful when we make AI play against each other. However, the clues can become incomprehensible for humans.

Instead, we propose a solution based on Quasimodo to generate clues. We consider that the words to guess are subjects. Then, we take the object associated with the more subjects that has a score above a certain threshold and does not appear for the wrong subjects.

In the demonstration scenario, the user plays the role of the operative. He receives clues and must click on the potential agents. He plays against a bot which simply guesses one right word per turn. Vlaada Chvátil suggests this strategy for games with two players.

5.6 Multiple-Choice Question Answering

As in the original paper, we added the possibility to perform question answering using only Quasimodo, i.e. we do not have an underlying language model. We used the same algorithm: Given a question, an answer and a knowledge base, we generate a set of features based on the connections between the words in the knowledge graph. Then, using the same training data as in Quasimodo, we train a linear classifier to predict a score for each answer. We reused the code provided with Quasimodo and added an interface to ask a question and give possible answers. Then, the system provides a score for each answer and displays them.

[9]https://sites.google.com/view/the-codenames-ai-competition

6 CONCLUSION

With this demonstration, we give insights into the commonsense knowledge base Quasimodo. The user can access in a user-friendly way to the raw data of Quasimodo. Besides, more details are given about the generation of the statements, making the entire process completely transparent. Finally, we showcased applications such as Taboo to prove the value of the database. Quasimodo is a scalable system to mine commonsense knowledge based on a general and adaptive extraction pipeline. We hope that this work will provide a better understanding of the system and the data so researchers can keep building on top of it, either on the application side or on the system side by proposing new extensions.

REFERENCES

[1] Antoine Bosselut, Hannah Rashkin, Maarten Sap, Chaitanya Malaviya, Asli Çelikyilmaz, and Yejin Choi. 2019. COMET: Commonsense Transformers for Automatic Knowledge Graph Construction. In *ACL*.

[2] Yohan Chalier, Simon Razniewski, and Gerhard Weikum. 2020. Joint Reasoning for Multi-Faceted Commonsense Knowledge. *AKBC* (2020).

[3] Bhavana Dalvi, Niket Tandon, and Peter Clark. 2017. Domain-Targeted, High Precision Knowledge Extraction. In *TACL*.

[4] Douglas B Lenat. 2019. What AI can learn from Romeo and Juliet. https://www.forbes.com/sites/cognitiveworld/2019/07/03/what-ai-can-learn-from-romeo--juliet/. Accessed: 2020-04-25.

[5] Douglas B Lenat, Mayank Prakash, and Mary Shepherd. 1985. CYC: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI magazine* (1985).

[6] Gary Marcus. 2020. The Next Decade in AI: Four Steps Towards Robust Artificial Intelligence.

[7] John McCarthy. 1960. *Programs with common sense.* RLE and MIT computation center.

[8] Koninika Pal, Vinh Thinh Ho, and Gerhard Weikum. 2020. Co-Clustering Triples from Open Information Extraction. In *CoDS-COMAD*.

[9] Thomas Pellissier Tanon, Gerhard Weikum, and Fabian M. Suchanek. 2020. YAGO 4: A Reason-able Knowledge Base. In *ESWC*.

[10] Julien Romero, Simon Razniewski, Koninika Pal, Jeff Z. Pan, Archit Sakhadeo, and Gerhard Weikum. 2019. Commonsense properties from query logs and question answering forums. In *CIKM*.

[11] Maarten Sap, Ronan LeBras, Emily Allaway, Chandra Bhagavatula, Nicholas Lourie, Hannah Rashkin, Brendan Roof, Noah A Smith, and Yejin Choi. 2018. Atomic: An atlas of machine commonsense for if-then reasoning. *AAAI* (2018).

[12] Robyn Speer and Catherine Havasi. 2012. ConceptNet 5: A large semantic network for relational knowledge. In *Theory and Applications of NLP*.

[13] Niket Tandon, Gerard de Melo, and Gerhard Weikum. 2017. WebChild 2.0: Fine-grained commonsense knowledge distillation. In *ACL*.