Towards Explainable Machine Learning Using Natural Language Processing and Ontologies

***Abstract*— We developed a software agent intended to introduce users in machine learning domain. We argue on the usefulness of a chatbot in the machine learning field. Our chatbot relies on structured knowledge on machine learning encapsulated as a domain ontology.**

I. Introduction

Searching for *machine learning* (ML) keywords has quadrupled in the last 5 years, reaching new highs in 2019, according to Google statistics. The growth of interest in machine learning and the development of this area is encouraged by the increase in the number of available data that can be used to train algorithms. Moreover, people are interested, but also skeptical about the results of an algorithm. This has led to the need for *Explainable AI* (XAI), to explain the process of obtaining the answer and gaining user trust.

The goal XAI is to support user with the following questions against a black-box systems (like machine learning): Why did you do that? Why not something else? When do you succeed/fail? When can I trust you? How to I correct an error? We are her eon the same line with the following quotation [1] from CACM: *"It's time for AI to move out its adolescent, game-playing phase and take seriously the notions of quality and reliability."*.

In the same line, EU policy towards data protection also advocates the right to explanation. Hence, there is no surprise that GDPR article 22 includes: *"The right not to be subject to a decision based solely on automated processing"*.

The topic of responsible data science [2] is a first example of a blind spot that can benefit from XAI. Our view is that ML alone is not enough for decision making. Even if with accurate classification you need a kind of guarantee or reasoning to take decisions.

Many approaches are available from the earliest years, such as Toni et al. work on argumentation and explanations [3]. Similarly, Gilpin et al. [4] focus on explainable methods in deep neural architectures, giving an overview of interpretability of machine learning. These achievements do not stop at explaining machine learning, but go on to various areas such as medicine, biology and natural language. In biomedical field, Holzinger et al. [5] hold the implementation of interactive machine learning, *iML*. This approach puts the human in the algorithmic loop, combining human knowledge with incomplete biomedical data to obtain what neither a human nor a computer could do on their own. It is understandable the need for *XAI*.

We have developed a software agent that transforms natural language questions into SPARQL, providing query editing and then running it to get the answer. It tries to be transparent, and to involve the user in the process of extracting the answer, explaining its provenance.

The rest of the paper is structured as follows: Section II details the architecture of the agent. Section III describes some types of questions that are handled by our agent. A discussion based on related work is found in section IV, while the section V concludes the paper.

II. System architecture

The developed system[1] has four modules depicted in Figure 1. First, the *user interface* (UI) is a web page where the user introduces questions in natural language related to the machine learning domain. These questions are automatically translated into SPARQL in order to obtain answers from an ontology for machine learning. Second, the *reasoning services* on description logics (DL) are provided by the Pellet library where a Spring Boot module is integrated. The Spring Boot module provides endpoints used through the UI. Third, the *translator* module is responsible for obtaining the SPARQL form of the query. Forth, *machine learning ontology* is the knowledge source formalising definitions and facts from the machine learning domain. These four modules are detailed as follows.

A. User Interface

In the main tab (see Fig. 5 from section III), the user can enter a question and gets a query in SPARQL. This syntax can be further modified by an expert users, if needed. The user obtains the answer querying the ML ontology with two reasoning tools: Pellet or RDFLib.

B. Reasoning services

Reasoning services in DL are performed by the Pellet reasoner[2] developed by Parsia et al. [6]. Pellet supports queries in SPARQL and infers knowledge that is not explicitly stated in the ML ontology.

The Spring Boot module acts as the server of the application. It provides a set of endpoints (see Table I) that can be called from the user interface. The responses of these endpoints are transmitted back to the user interface.

C. Translating from natural language to SPARQL

We use Quepy framework to translate natural language question to SPARQL or MQL. Quepy relies on $refo$[3], $nltk$[4] for natural language tagging and $SPARQLWrapper$[5] for

[1] Available at https://bitbucket.org/nicoleta_corocea/ml_chatbot/src/master/
[2] https://github.com/stardog-union/pellet
[3] https://github.com/machinalis/refo
[4] http://www.nltk.org/
[5] https://pypi.org/project/SPARQLWrapper

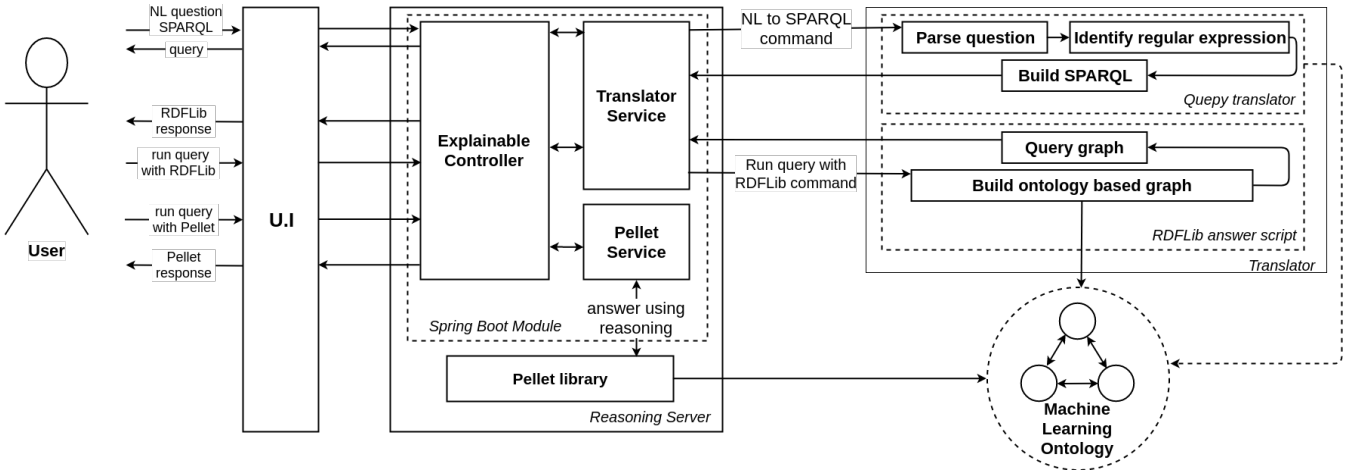

Fig. 1. System architecture.

TABLE I
REASONING SERVICE ENDPOINTS.

Endpoint	Usage
translateQuery	translate query from NL language to SPARQL
runQueryWithPellet	run SPARQL query using Pellet reasoner
runQueryWithRDFLib	run SPARQL query using RDFLib
getconcepts	get concepts of the ML ontology using Pellet
getProperties	get properties of the ML ontology using Pellet
getIndividuals	get individuals of the ML ontology using Pellet

working with SPARQL. We customized Quepy to match different types of questions. Regexes were added to match questions.

We needed to modify Quepy to avoid some of its limitations. For instance, we added support for filter and logic operators (AND, OR, NOT), which often appear in natural language queries.

Various steps are applied to a query in NL in order to obtain its SPARQL syntax (see Algorithm 1). Here, $\mathcal{Q}$ is the set of question template concepts declared for parsing purposes. $\mathcal{R}$ is the set of regular expressions collected from question template concepts. W_q represents the set of words in query q. $graph$ is structure built by interpreting the order of matched words.

Fig. 2 details the rule R_1. Here $target$ keeps the place of the concept we want to display the individuals in the ontology (e.g. $Algorithm$, $Data$). It can be a noun (NN), proper noun (NNP), or plural of a noun (NNS). The $optOpening$ stands for *"What are the"* or *"Which are the"* and it is optional by $Question$ function's behaviour. By $Lemma("be")$, we understand the base word and its inflections (i.e. all of the forms of *to be*). The expressions $regex1$ and $regex2$ are joined under $regex$ and represent the accepted forms of the question.

Example 1 (Applying R_1): Let the query *What are the individuals of Algorithm?* The query matches on $regex1$ as follows:

Data: Query q in natural language
Result: SPARQL query
$\mathcal{R} \leftarrow extractRules(\mathcal{Q})$;
$W_q \leftarrow tokenize(q)$;
$q_{SPARQL} =$ " ";
foreach $r \in \mathcal{R}$ **do**
  if $match(r, \mathcal{W}_q)$ **then**
    $className = provenanceClass(r)$;
    $graph[srcNode, (property, destNode)] =$
     $className.interpret(r, \mathcal{W}_q)$;
    $q_{SPARQL} = graphToSparql(graph)$
  end
  return q_{SPARQL}
end

Algorithm 1: Processing flow of a query using Quepy.

```
1  target = Group (Pos("NN") |
2                  Pos("NNP") |
3                  Pos("NNS"), "target")
4  opt_opening = Question((Pos("WP") |
5                          Pos("WDT")) +
6                            Lemma("be") +
7                            Pos("DT"))
8  regex1 = optOpening +
9           Question(Lemma("individual") |
10                   Lemma("instance")) +
11                     Pos("IN") + target +
12                     Question(Pos(".")))
13 regex2 = Question(Lemma("list") |
14                   Lemma("show") |
15                   Lemma("print")) + target +
16                    Question(Lemma("individual") |
17                             Lemma("instance"))
18 regex = regex1 | regex2
```

Fig. 2. Rule R_1 for listing individuals.

TABLE II

SAMPLE OF PARSING RULES.

$\mathcal{R}$	*Expression*
R_2	given algorithm has learning method
R_3	given algorithm solves learning problem AND has learning method
R_4	given algorithm is suitable for a data characteristic OR another
R_5	given algorithm does NOT have learning method
R_6	list parameters of a given algorithm
R_7	superconcepts of an instance
R_8	the comment of an individual

regex part	word
Pos("WP")	What
Lemma("be")	are
Pos("DT")	the
Lemma("individual")	individuals
Pos("IN")	of
target	Algorithm

Other parsing rules similar to R_1 are listed in Table II.

1) RDFLib: To handle situations in which Quepy is not working with Pellet, we rely also on the RDFLib[6]. RDFLib stores the ontology as an RDF graph, which is a set of RDF triples. The set of nodes of an RDF graph is the set of subjects and objects of triples in the graph. Unlike Pellet, RDFLib is not a reasoner, therefore it will not perform reasoning on the ontology and it will not infer new axioms. We use RDFLib as an alternative to Pellet, to compare the results and to observe the differences. The user has the possibility to run the query using RDFLib, but most of the times, Pellet suffices.

D. Machine learning ontology

Aware of the difficulties of building a complete ontology for machine learning, we developed here only a prototype to illustrate the functionalities of our system. Our machine learning ontology formalises knowledge for 340 instances, 56 concepts and 9 roles from the machine learning domain. A view of this ontology appear in 3. One can see here the roles that link the ML *Algorithm* concept with other concepts from the ontology. The ontology contains data on algorithms, their parameters, learning method, solved learning problem, metric performance, data characteristics and features, advantages and disadvantages, and also information on their provenance.

The *Algorithm* concept is divided into several subconcepts that represent different families of algorithms, as can be seen in Fig. 4. It brings together algorithms that have supervised, semi-supervised, unsupervised (as clustering and association) or reinforcement learning methods. Similar to the *Algorithm* concept, the parameters are grouped into the *Parameter* concept by the algorithms to which they are associated.

Ontology was developed in Protege [7], which provides support for both Pellet and SPARQL queries. For ontology population we rely on various tools available for ML. For clustering, semi-supervised and supervised algorithms the main source of information was SciKit[8]. For association mining, we used spmf[9]. Data on reinforcement learning comes from UNSW[10].

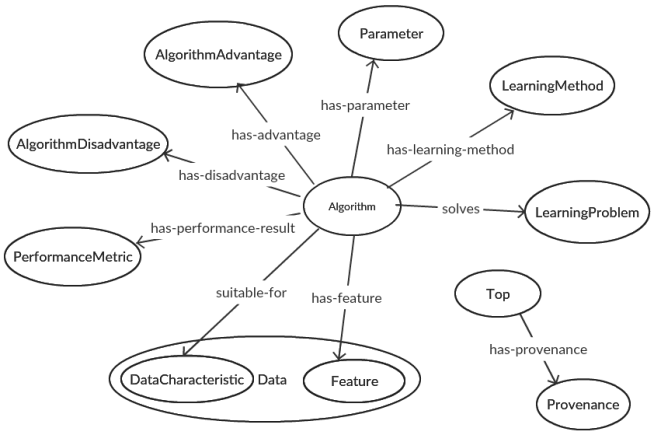

Fig. 3. View on the ML ontology: the *Algorithm* concept and its roles with other concepts.

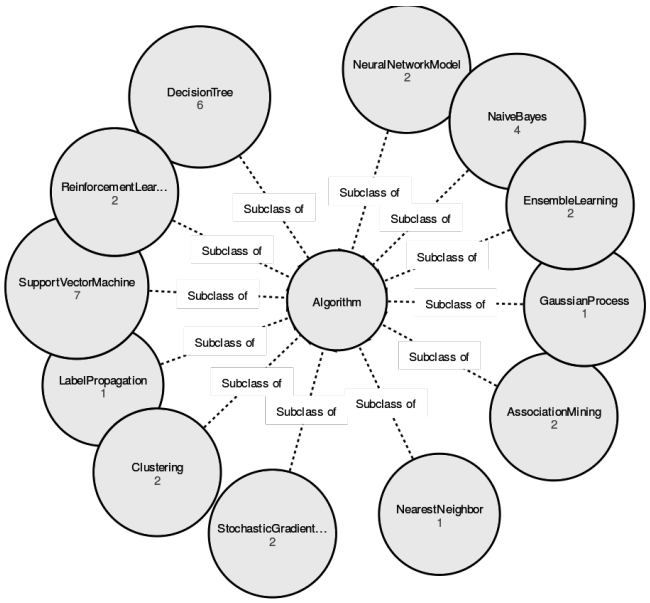

Fig. 4. View on the machine learning ontology: various ML algorithms.

III. RUNNING SCENARIO

This section illustrates the capabilities of our system through a running scenario. Assume the user starts with the following question:

Q_1: Which are the instances of algorithms?

This question can also be formulated as *"individuals of Algorithm"*, *"print Algorithm instances"*, *"show Algorithms individuals"*, *"list Algorithm"* and it will be processed in the same way. The word "individual" can be replaced with

[6]https://rdflib.readthedocs.io/en/stable/

[7]https://protege.stanford.edu/

[8]https://scikit-learn.org/stable/

[9]http://www.philippe-fournier-viger.com/spmf

[10]https://www.cse.unsw.edu.au/∼cs9417ml/RL1/algorithms.html

"instance", but it may also be missing. Given that the plural of a concept can be a common usage, we also considered this case. Based on the Quepy rule R_1 (formalised in section II) the corresponding SPARQL for Q_1 appears in Listing 1:

```
PREFIX rdf:<http://www.w3.org/1999/02/
          22-rdf-syntax-ns#>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
PREFIX quepy:<http://www.machinalis.com/quepy#>
PREFIX ml:<http://www.semanticweb.org/
          machine-learning-ontology#>

SELECT DISTINCT ?x0 WHERE {
  ?x0 rdf:type ml:Algorithm.
}
```

Listing 1. SPARQL formalisation for the query Q_1.

Using Pellet, the answer a_1 is

```
{"head": { "vars": [ "x0" ]} ,
  "results": {
    "bindings": [
      {"x0":{"value":"gaussianProcessRegressor"}},
      {"x0":{"value":"oneClassSVM" }}
      ...
      {"x0": {"value":"decisionTreeClassifier"}}]}}
```

where the variable $?x0$ stores the instances of the Algorithm concept. The answer is obtained based on the following general inclusion axioms in description logic:

1. $GaussianProcess \sqsubseteq Algorithm$
2. $SupportVectorMachine \sqsubseteq Algorithm$
3. $DecisionTree \sqsubseteq Algorithm$
4. $gaussianProcessRegressor : GaussianProcess$
5. $oneClassSVM : SupportVectorMachine$
6. $decisionTreeClassifier : DecisionTree$

Pellet deduces from the axioms (1) and (2) that $GaussianProcess$, $DecisionTree$ and $SupportVectorMachine$ are subconcepts of $Algorithm$. Given axioms (4), (5) and (6), Pellet infers that $gaussianProcessRegressor$, $oneCLassSVM$ and $decisionTreeClassifier$ are also individuals in the concept $Algorithm$.

Differently, using RDFLib, the answer is an empty list because RDFLib can't perform reasoning on given facts and it can't deduce that $gaussianProcessRegressor$, $oneCLassSVM$ and $decisionTreeClassifier$ are individuals of $Algorithm$. In this case, if we wanted $DecisionTree$'s instances then RDFLib would have responded correctly with a list of individuals. For example, given the query in Listing 2 in which we ask for individuals of the $DecisionTree$ concept:

```
SELECT DISTINCT ?x0 WHERE {
  ?x0 rdf:type ml:DecisionTree.}
```

Listing 2. SPARQL query for obtaining the instances of the DecisionTree concept.

The answer a_2 computed by RDFLib is:

```
[{ "x0": "decisionTreeRegressor"},
...
{"x0": "decisionTreeClassifier"},
{"x0": "id3"}]
```

Assume that the user continues with question Q_2:

Q_2: What are the learning methods of linearSVR?

This question can also conveyed as *"linearSVR learning methods"*, *"What are the learning methods of linearSVR?"*, *"print linearSVR learning methods"*, *"list linearSVR learning methods"*. Using Quepy rule R_2, the obtained SPARQL query appears in Listing 3.

```
SELECT DISTINCT ?x1 WHERE {
  ?x0 rdf:type ml:Algorithm.
  FILTER(?x0 = ml:linearSVR).
  ?x0 ml:hasLearningMethod ?x1.}
```

Listing 3. SPARQL version for the query Q_2.

Here, the variable $?x0$ stores the individual of $Algorithm$ whose name is $linearSVR$. The variable $?x1$ holds the place of an individual to whom $?x0$ has $ml : hasLearningMethod$ property. Using Pellet, the answer a_3 is

```
{"x1": { "value": "supervised" }}
```

The results in a_3 are deduced on the axioms (6)-(9):

6. $SupportVectorMachine \sqsubseteq Algorithm$
7. $linearSVR : SupportVectorMachine$
8. $supervised : LearningMethod$
9. $linearSVR\ hasLearningMethod\ supervised$

Using RDFLib, the answer is empty, because it can't perform reasoning, and thus it can not identify $linearSVR$ as instance of $Algorithm$.

Next, assume the user requests informations using Q_3:

Q_3 *Algorithm that solves classification problem and has supervised learning method and has feature complexInteractions algorithm that solves classification and has supervised and complex interactions.*

The query is matched against the Quepy rule R_3. This leads to the SPARQL query in Listing 4. Here $?x0$ is an instance of $Algorithm$. $?x1$ stores $supervised$ individual of $LearningMethod$, $?x2$ stores $classification$ of $LearningProblem$ and $?x3$ holds the place for $complexInteractions$ which is of type $Feature$.

The answer a_4 given by Pellet is:

```
{"x0": { "value": "decisionTreeClassifier" }}
```

```
SELECT DISTINCT ?x0 WHERE {
  ?x0 rdf:type ml:Algorithm.
  ?x0 ml:has-learning-method ?x1.
  ?x0 ml:solves ?x2.
  ?x0 ml:has-feature-characteristic ?x3.
  ?x1 rdf:type ml:LearningMethod.
  FILTER(?x1 = ml:supervised
        && ?x2 = ml:classification
        && ?x3 = ml:complexInteractions).
  ?x2 rdf:type ml:LearningProblem.
  ?x3 rdf:type ml:Feature.}
```

Listing 4. SPARQL query for the Q_3 question.

10. $DecisionTree \sqsubseteq Algorithm$
11. $dtClassifier : DecisionTree$
12. $supervised : LearningMethod$
13. $classification : LearningProblem$
14. $complexInteractions : Feature$
15. $(dtClassifier, supervised) : hasLearningMethod$
16. $(dtClassifier, classification) : solves$
17. $(dtClassifier, complexInteractions) : hasFeatureCharacteristic$

The result is obtained based on the axioms (10)-(17). For instance $decisionTreeClassifier$ is identified as an instance of $Algorithm$. Based on axioms (15)-(17), it is selected as the right result by Pellet. Instead, RDFLib is not able to identify $decisionTreeClassifier$ as a instance of $Algorithm$.

Asume the user requests further information on algorithms suitable for non linear models, as in Q_4:

Q_4 *Algorithms suitable for nonLinearModels or shortDocuments data characteristics.*

This query can be written as *"algorithms suitable for nonLinearModels or shortDocuments"*, leading to the same result. Using the Quepy rule R_4, the corresponding SPARQL appears in Listing 5. Here, the variable $?x0$ stores the individual of $Algorithm$. The variable $?x1$ holds the place of an individual of $DataCharacteristic$ which is $nonLinearModels$ or $shortDocs$.

```
SELECT DISTINCT ?x0 WHERE {
 ?x0 rdf:type ml:Algorithm.
 ?x0 ml:suitable-for ?x1.
 ?x0 ml:suitable-for ?x2.
 ?x1 rdf:type ml:DataCharacteristic.
 FILTER(?x1 IN ( ml:nonLinearModels, ml:shortDocs)).
 ?x2 rdf:type ml:DataCharacteristic.}
```

Listing 5. SPARQL query version for the query Q_4

Using Pellet, the answer a_5 is

```
[{"x0": { "value": "mlpClassifier" }} ,
{"x0": { "value": "mlpRegressor" }} ,
{"x0": { "value": "decisionTreeClassifier" }} ,
{"x0": { "value": "bernoulliNaiveBayes" }}]
```

The axioms used for $mlpClassifier$ and $bernoulliNaiveBayes$ are:

18. $NeuralNetworkModel \sqsubseteq Algorithm$
19. $NaiveBayes \sqsubseteq Algorithm$
20. $mlpClassifier : NeuralNetworkModel$
21. $bernoulliNaiveBayes : NaiveBayes$
22. $nonLinearModels : DataCharacteristic$
23. $shortDocs : DataCharacteristic$
24. $(mlpClassifier, nonLinearModels) : suitableFor$
25. $(bernoulliNaiveBayes, shortDocs) : suitableFor$

$DataCharacteristics$ are $nonLinearModels$ and $shortDocs$. Hence, some instances of $Algorithm$ are required to satisfy the $suitableFor$ property. Pellet identifies $mlpClassifier$ and $bernoulliNaiveBayes$ as the needed instances of $Algorithm$ based on axioms (18)-(21) and (24)-(25).

In the following utterance, the user is interested in something different of supervised learning:

Q_5 *Algorithms that do not have supervised learning method.*

Writing this query as *"algorithm that does not have supervised"* will lead to the same result. R_5 is the Quepy rule for the corresponding SPARQL in listing 6. The variable $?x0$ stores the individual of $Algorithm$. The variable $?x1$ holds the place of $supervised$ which has the type $LearningMethod$. $?x0$ should not have $?x1$ learning method. Note here that we used the NOT operator that we introduce on top of Quepy.

```
SELECT DISTINCT ?x0 WHERE {
  ?x0 rdf:type ml:Algorithm.
  MINUS { ?x0  ml:has-learning-method ?x1.
FILTER(?x1 = ml:supervised).}.
  ?x1 rdf:type  ml:LearningMethod.}
```

Listing 6. SPARQL query version for the query Q_5

The answer a_6 given by Pellet is

```
[{"x0": { "value": "oneClassSVM" }},
 {"x0": { "value": "agglomerativeClustering" }},
 ...
 {"x0": { "value": "labelPropagation" }}]
```

The axioms used for $labelPropagation$ are:

26. $LabelPropagation \sqsubseteq Algorithm$
27. $labelPropagation : LabelPropagation$
28. $supervised : LearningMethod$
29. $semiSuperviseds : LearningMethod$
30. $(labelPropagation, semiSupervised) : hasLearningMethod$

The individual $labelPropagation$ is of type $LabelPropagation$ and, given axioms (26)-(27), is of type $Algorithm$. Given the current SPARQL query we want instances of $Algorithm$ that do not have $supervised$ learning method. With the help of (29)-(30)

$labelPropagation$ is identified as a correct response having $semiSupervised$ learning method.

Assume the user is interested in specific details about the Bernoully Naive Bayes:

Q_6 *What are the parameters of bernoulliNaiveBayes?*

The query can be written as *"list bernoulliNaiveBayes parameters"*, *"print bernoulliNaiveBayes parameters"*, *"bernoulliNaiveBayes parameters"* also, leading to the same result.

Using the Quepy rule R_6, the question Q_6 is translated SPARQL in Listing 7). The variable $?x0$ stands for $bernoulliNaiveBayes$ individual of $Algorithm$. $?x0$ has $?x1$ parameter.

```
SELECT DISTINCT ?x1 WHERE {
  ?x0 rdf:type ml:Algorithm.
  FILTER(?x0 = ml:bernoulliNaiveBayes).
         ?x0 ml:has-parameter ?x1.}
```

Listing 7. SPARQL query for Q_6.

The answer given by Pellet a_7 is

```
[{"x1": { "value": "fit_prior" }} ,
 {"x1": { "value": "class_prior" }} ,
 {"x1": { "value": "alpha" }} ,
 {"x1": { "value": "binarize" }}]
```

The axioms used to compute the answer for $fitPrior$ are:

31. $NaiveBayes \sqsubseteq Algorithm$
32. $bernoulliNaiveBayes : NaiveBayes$
33. $NaiveBayesParameter : Parameter$
34. $BernoulliNaiveBayesParameter : NaiveBayesParameter$
35. $fitPrior : BernoulliNaiveBayesParameter$
36. $(bernoulliNaiveBayes, fitPrior) : hasParameter$

Based on axioms 31-32, the individual $bernoulliNaiveBayes$ is an $Algorithm$. The individual fit_prior is of type $BernoulliNaiveBayesParameter$ and, given axioms (32)-(36), is identified as a correct response and thus it is added in response list.

The user can compare also various algorithms:

Q_7 *Compare sarsa and labelPropagation.*

This type of question is a special case due to the fact that the SPARQL query is not generated by Quepy but is built in the Spring Boot module. It can be one out of three types: by learning method, by the solved learning problem, by parameters of the two algorithms. The correponding questions are: *Compare sarsa and labelPropagation by learning method*, *Compare sarsa and labelPropagation by solved problem*, *Compare sarsa and labelPropagation by algorithm parameters*. We will continue to describe the first question, *Compare sarsa and labelPropagation by learning method*, the other being similar. The formalisation in SPARQL appears in

In this query, the variable $?x1$ corresponds to $sarsa$ or $labelPropagation$ individuals, while $?x0$ is an individual to whom $?x1$ has a learning-method property. Using Pellet, the answer a_8 is:

```
SELECT DISTINCT ?x1 ?x0 WHERE {
    ?x1 rdf:type ml:Algorithm.
    ?x1 ml:has-learning-method ?x0.
    FILTER(?x1 IN (ml:sarsa, ml:labelPropagation)).
}
```

Listing 8. SPARQL query

```
[{"x1": { "value": "sarsa" } ,
   "x0": { "value": "reinforcement" }},
 {"x1": { "value": "labelPropagation" },
  "x0": { "value": "semiSupervised" }}]
```

The axioms used for this response are:

37. $ReinforcementLearning \sqsubseteq Algorithm$
38. $LabelPropagation \sqsubseteq Algorithm$
39. $sarsa : ReinforcementLearning$
40. $labelPropagation : LabelPropagation$
41. $reinforcement : LearningMethod$
42. $semiSupervised : LearningMethod$
43. $(sarsa, reinforcement) : hasLearningMethod$
44. $(labelPropagation, semiSupervised) : hasLearningMethod$

Using the axioms (37)-(40), $sarsa$ and $labelPropagation$ are identified as $Algorithms$. By axioms (41)-(44), the algorithms are linked to their corresponding learning methods.

Another set of questions handled by our system is represented by question Q_8:

$\{Q_8$*: What is max_depth*

This question can also be formulated as *"max_depth type"*, *"max_depth class"*,*"type of max_depth"*, *"class of max_depth"* and it will find the parent concepts to which it belongs.

Using the Quepy rule R_8, the corresponding SPARQL appears in Listing 9. Here $?x0$ stands for the individual whose name is max_depth. $?x1$ holds the place for $?x1$'s type.

```
SELECT DISTINCT ?x1 WHERE {
  FILTER(?x0 = ml:max_depth).
  ?x0 rdf:type ?x1.}
```

Listing 9. SPARQL form for the query Q_8.

Pellet infers the following answer a_9:

```
[{"x1": { "value": "EnsembleMethodsParameter" }} ,
 {"x1": { "value": "Thing" } } ,
 {"x1": { "value": "Parameter" }} ,
 {"x1": { "value": "RandomForestParameter" }} ,
 {"x1": { "value": "DTParameter" }}]
```

The answer is obtained based on the following general inclusion axioms in description logic:

45. $Parameter \sqsubseteq \top$
46. $EnsembleMethodPar \sqsubseteq Parameter$
47. $RandomForestPar \sqsubseteq EnsembleMethodPar$
48. $DTPar \sqsubseteq Parameter$
49. $maxDepth : RandomForestPar$
50. $maxDepth : DTPar$

Pellet has inferred the entire tree of the concepts and subconcepts to which $maxDepth$ belongs. max_depth is an instance of $DTPaR$, which is the subconcept of $Parameter$. In its turn, $Parameter$ is a subconcept of $\top$.

Differently, using RDFLib, the answer a'_9 is:

```
[{"x1": "NamedIndividual" },
 {"x1": "RandomForestParameter"},
 {"x1": "DTParameter"}]
```

RDFLib recognizes only direct ancestors, as can be seen in 51-53 axioms:

51. $maxDepth : NamedIndividual$
52. $maxDepth : RandomForestParameter$
53. $maxDepth : DTParameter$

A final feature exemplified here appears in the open question Q_9*: Tell me more about minconf.*

Other forms of this question are *"show details of minconf"*, *"list definitions of minconf"*,*"print definition of minconf"*, *"define minconf individual"*. All of these forms will find the comments associated with the individual $minconf$. Using the Quepy rule R_9 the corresponding SPARQL appears in Listing 10. Here, $?x0$ stands for the individual whose name is $minconf$. $?x1$ stands for the node that represents $minconf$ individual. $?x2$ holds the place for $?x0$'s comment.

```
SELECT DISTINCT ?x2 WHERE {
  FILTER(?x0 = ml:minconf).
  ?x1 owl:annotatedSource ?x0.
  ?x1 rdfs:comment ?x2.}
```

Listing 10. SPARQL translation of the question Q_9.

Both Pellet and RDFLib return the answer a_{11}:

```
[{"x2": { "value":
  "[Association] minimum threshold of confidence"}}]
```

The answer is obtained based on axioms:

54. $minconf : Parameter$
55. $(b0, minconf) : annotatedSource$
56. $(b0, "[Association]\ minimum\ threshold\ of\ confidence") : comment$

Each individual is represented by a unique node, one for every concept it belongs to. Let $b0$ in our case, as axiom (55) shows. The comment is linked to that node. An individual can have a different comment for every concept it belongs to.

In Figure 5 it is represented how the user uses the interface to get answers as illustrated with Q_1 - Q_9.

Fig. 5. The online system. Asking a query in natural language about machine learning

IV. Discussion and related work

Goel has built a generation of teaching assistants [7] that are able to answer frequently asked questions on the online question forum of its course. The first out of the three teaching assistants is based on IBM Watson. The other two are based on a software developed by Goel and his colleagues. JW1 (Jill Watson, the first) is a memory of question-answer pairs extracted from the existing data on the forum. The pairs are organized into categories by questions. Unlike his ancestor, JW2 doesn't have episodic memory. JW2 uses semantic processing based on conceptual representations. It maps the introduction of the question into relevant concepts and then uses concepts to retrieve a precompiled response.

Karale et al. have developed a retrieval system based on an ontology [8]. It uses Quepy as natural language interpretor to translate user's questions to SPARQL queries. The query is fired to RDF ontology and RDFLib is used to retrieve the answer. The system uses WordNet to find semantic query terms in order to provide query expansion.

Our work is in line with Goel's generation of teaching assistants. We restricted here to natural language questions about machine learning addressed to an agent which will give the right answers. Along the way, we decided that the query language of the ontology would be SPARQL, thanks to the support provided by Protege. The retrieval system of Karale et al. is in the same line as our, encouraging the use of the Quepy framework.

Several translators from natural language to SPARQL exist.

AutoSPARQL [9] allows to answer natural language queries over RDF ontologies. It uses active supervised machine learning algorithms from DL-Learner Semantic Web machine learning framework to generate a SPARQL query. The algorithm and the results are improved by answering to questions and deciding whether an entity belongs to the result set. It can generate complex queries than most knowledge base specific applications.

QuestIO [10] translate natural language queries into a set of SPARQL or SeRQL queries. The set is ranked and the queries are executed until a valid set of results is found. First, the query is interpreted and the key concepts and keywords

are identified. The key concepts are then analysed and the system infers potential relations among these concepts. The relations are ranked by similarity and position in ontology and are transformed into SPARQL or SeRQL.

Quepy transforms natural language questions to SPARQL or MQL. It can be customized to match different types of questions. Regular expressions are defined and those will match natural language questions and transform them into an abstract semantic representation.

For justifying our preference among Quepy, AutoSPARQL and QuestIO, we have two arguments: First, we had difficulties to integrate AutoSPARQL with our machine learning ontology. Second, between QuestIO and Quepy, we chose Quepy due to its better documentation.

Similarly, we analysed various reasoners on description logics to incorporate in our system.

Haarslev et al. [11] have developed Racer that also can be accessed by HTTP or TCP protocols. It can read OWL ontologies and it can receive instructions or queries from users via TCP protocol. Hermit [12] supports several specialised reasoning services, as well as SPARQL queries, SWRL rules and Description Logic safe rules. It uses optimisations to ensure efficient processing of real-world ontologies. Pellet [13] has a built-in module that support queries written in SPARQL language, as well as in RDQL query language.

The first option considered was the use of Racer. To make SPARQL queries on the ontology, a RDF triple store managed by AllegroGraph [14] system should be used [15]. We did not manage to start this feature on the public version of the system. On the other hand, Hermit does not include examples of use and does not provide documentation for the methods that can be used. We chose Pellet because it was easier to integrate, a major advantage being the examples provided in which SPARQL language was used.

During engineering our machine learning ontology, we tried to reuse different knowledge sources:

ML-Schema [16] is a top-level ontology to interchange information on machine learning algorithms, data sets, and experiments. It is developed based on the existing ontologies: Expose, DMOP, MEX, OntoDM-core.

Expose ontology [17] describes machine learning experiments and analysis of learning algorithms. It is designed to collect and organize details of experiments. It is described in OWL-DL language and is focused on supervised classification on propositional data sets.

DMOP [18] is a Data Mining Optimization Ontology Schema that aims at algorithm automation and model selection. It contains descriptions of data mining task, data and algorithms.

MEX Vocabulary [19] was developed to fulfill the gap given by missing a specification for interchanging machine learning metadata between different architectures. It is composed by: i) MEX-Core - contains the key entities in machine learning; ii) MEX-Algorithm - represents the context of machine learning algorithms and their characteristics; and iii) MEX-Performance - supplies basic entities to represent the experimental results of machine learning algorithms.

Onto-DM [20] has been designed to include a representation of data mining field. It covers basic data mining definitions, data mining algorithms and complex entities based on constraints. It is build using the best practices of ontology engineering.

From this ML-Schema, we have reused the data characteristics concepts: Data and its subconcepts, DataCharaceristic and Feature. Also, from MEX-Algorithm we used the structure for LearningMethod and LearningProblem concepts, and we transformed their subconcepts in individuals. Expose, DMOP, Onto-DM have not provided information or schema-parts relevant to the line in which we want to drive our ontology.

V. Conclusion

Our system aims to explain machine learning with natural language, SPARQL queries and ontologies. Quepy was used to transform natural language queries into SPARQL. Using the Pellet reasoner, or the RDFLib representation as a graph, the user can question the machine learning ontology. We improve Quepy in directions of logical operators (not, and, or) and for filtering by the individual's name.

The challenge remains to engineer a complete machine learning ontology, empowered with automatic means of enrichment, as machine learning is a dynamic domain. We are currently developing parsing rules for the *frequent asked questions* on machine learning as they appear in the public arena.

The current work can be integrated into the larger context of Explainable AI. Our view is that sooner or later, legislation will force some software products based on machine learning to have a mandatory *Explain me!* button.

References

[1] D. Monroe, "Ai, explain yourself," *Communications of the ACM*, vol. 61, no. 11, pp. 11–13, 2018.

[2] W. M. van der Aalst, M. Bichler, and A. Heinzl, "Responsible data science," 2017.

[3] K. Cyras, O. Cocarascu, and F. Toni, "Explanatory predictions with artificial neural networks and argumentation," 07 2018.

[4] L. H. Gilpin, D. Bau, B. Z. Yuan, A. Bajwa, M. Specter, and L. Kagal, "Explaining Explanations: An Overview of Interpretability of Machine Learning," Tech. Rep., 2019. [Online]. Available: https://arxiv.org/pdf/1806.00069.pdf

[5] A. Holzinger, "From Machine Learning to Explainable AI," in *2018 World Symposium on Digital Intelligence for Systems and Machines (DISA)*. IEEE, 8 2018, pp. 55–66. [Online]. Available: https://ieeexplore.ieee.org/document/8490530/

[6] E. Sirin and B. Parsia, "Sparql-dl: Sparql query for owl-dl." in *OWLED*, vol. 258. Citeseer, 2007.

[7] J. Watson, A. K. Goel, and L. Polepeddi, "A Virtual Teaching Assistant for Online Education," Tech. Rep. [Online]. Available: https://www.class-central.com/report/mooc-stats-2016/

[8] M. Pratibha, S. Sonakneware, and S. J. Karale, "Ontology Based Approach for Domain Specific Semantic Information Retrieval System," Tech. Rep., 2014.

[9] J. Lehmann and L. Bühmann, "AutoSPARQL: Let Users Query Your Knowledge Base," Tech. Rep. [Online]. Available: http://dbpedia.neofonie.de,

[10] "QuestIO - Natural Language Interfaces." [Online]. Available: https://sites.google.com/site/naturallanguageinterfaces/questio

[11] P. Hitzler, U. Sattler, V. Haarslev, K. Hidde, R. Möller, and M. Wessel, "The RacerPro Knowledge Representation and Reasoning System 1," Tech. Rep., 2011.

[12] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, "Hermit: an owl 2 reasoner," *Journal of Automated Reasoning*, vol. 53, no. 3, pp. 245–269, 2014.
[13] E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz, "Pellet: A Practical OWL-DL Reasoner," Tech. Rep. [Online]. Available: http://www.mindswap.org/2003/pellet/download.shtml
[14] "AllegroGraph - Semantic Graph Database." [Online]. Available: https://franz.com/agraph/allegrograph/
[15] "RacerPro User's Guide Version 2.0," Tech. Rep., 2012. [Online]. Available: http://www.racer-systems.com
[16] G. Correa Publio, D. Esteves, A. Ławrynowicz, P. Panče Panov, T. Soru, J. Vanschoren, and H. Zafar, "ML-Schema: Exposing the Semantics of Machine Learning with Schemas and Ontologies," Tech. Rep. [Online]. Available: http://ml-schema.github.io/documentation/
[17] "(PDF) Exposé: An ontology for data mining experiments."
[18] C. M. Keet, A. Ławrynowicz, C. D'amato, A. Kalousis, P. Nguyen, R. Palma, R. Stevens, and M. Hilario, "The Data Mining OPtimization Ontology," Tech. Rep. [Online]. Available: http://protege.stanford.edu
[19] D. Esteves, D. Moussallem, C. B. Neto, T. Soru, R. Usbeck, M. Ackermann, and J. Lehmann, "MEX vocabulary," in *Proceedings of the 11th International Conference on Semantic Systems - SEMANTICS '15*. New York, New York, USA: ACM Press, 2015, pp. 169–176. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2814864.2814883
[20] P. Panov, S. Deroski, and L. Soldatova, "OntoDM: An Ontology of Data Mining," in *2008 IEEE International Conference on Data Mining Workshops*. IEEE, 12 2008, pp. 752–760. [Online]. Available: http://ieeexplore.ieee.org/document/4734003/