A framework for syntactic and semantic quality evaluation of ontologies

Vivek Iyer1,2[0000−0003−4451−8293], Lalit Mohan Sanagavarapu1,2[0000−0003−0745−1042], and Y Raghu Reddy1,3[0000−0003−2280−5400]  
1 IIIT Hyderabad, India  
2 {vivek.iyer,lalit.mohan}@research.iiit.ac.in  
3 raghu.reddy@iiit.ac.in

Abstract. The increasing focus on Web 3.0 is leading to automated creation and enrichment of ontologies and other linked datasets. Alongside automation, quality evaluation of enriched ontologies can impact software reliability and reuse. Current quality evaluation approaches oftentimes seek to evaluate ontologies in either syntactic (degree of following ontology development guidelines) or semantic (degree of semantic validity of enriched concepts/relations) aspects. This paper proposes an ontology quality evaluation framework consisting of: (a) SynEvaluator and (b) SemValidator for evaluating syntactic and semantic aspects of ontologies respectively. SynEvaluator allows dynamic task-specific creation and updation of syntactic rules at run-time without any need for programming. SemValidator uses Twitter-based expertise of validators for semantic evaluation. The efficacy and validity of the framework is shown empirically on multiple ontologies.

Keywords: Ontology Quality Evaluation · Syntactic Evaluation · Semantic Validation · Crowdsourcing · Twitter-based expertise

1 Introduction

The exponential increase in Internet users over the past decade has led to generation of large volume of data. Web 3.0, otherwise commonly referred to as Semantic web, seeks to represent internet data as knowledge through knowledge graphs, ontologies and other knowledge systems [4]. These representations enable knowledge integration, semantic ambiguity resolution, information extraction, decision making, reasoning and many other use cases relevant to the building of ‘intelligent’ software systems. Ontologies, in particular, store domain-specific knowledge, and represent this knowledge through concepts, relations, axioms and instances. They contain a formal structure and achieve a certain level of rigor due to the presence of rules and constraints. Ontologies are rarely static in nature. The range and the depth of the knowledge stored are enriched over time. This impacts a wide variety of software applications that utilize ontologies for reasoning, decision-making, question-answering, etc. Ontology enrichment is thus a crucial step in the ontology engineering process.

Traditionally, ontologies are created and managed by knowledge engineers and domain experts resulting in high costs due to the expert human labour involved. Automated or semi-automated approaches to ontology enrichment are increasingly popular, driven by increased availability of domain-relevant internet data and improvements in natural language processing and machine learning models [36]. Research on ontology learning (both creation and enrichment) snowballed in the last two decades [20], [36], [15], with increased focus on fully automated Deep Learning based approaches [30]. Given the variety in ontology enrichment approaches, it is important to evaluate the quality of the enriched ontology.
Ontology evaluation approaches can broadly be divided into: manual, automated and semi-automated approaches. In general, ontology evaluation happens on one of two aspects: syntactic quality, or semantic quality. We define syntactic quality of an ontology as a measure of its adherence to ontology development guidelines or rules. For example, One such rule could necessitate presence of both domain and range elements in properties. Examples of other rules or guidelines could include explicit declaration of equivalent and inverse properties, presence of annotations \[34\], following of unique naming conventions \[23\], etc. Figure 1 shows, in Turtle syntax \[3\], a property without a defined range element - thus violating the rule that necessitates the presence of both domain and range elements. Semantic quality deals with validity of enriched concepts, relations and instances. Figure 2 shows an example ontology enriched with concepts extracted from a sentence to emphasize the need for evaluation of enriched ontologies. The previously existing concepts are shown in blue, the valid enriched concepts in green and the invalid enriched ones in red. Ontology enrichment algorithms using Hearst patterns \[14\] could mistakenly detect ‘antivirus’ as a type of ‘malware’. In such cases, before creating a final ontology, the enriched ontology needs to be validated for semantic quality, to accept or reject the enriched concepts, properties and instances.

A variety of metric-based methods were proposed to evaluate various syntactic quality-based characteristics of ontologies \[9\], \[19\], \[32\]. Publicly available tools were proposed that allow users to evaluate syntactical quality of ontologies using pre-defined metrics \[18\], \[22\] or rules \[28\], \[31\]. However, they offer limited customization and flexibility to the user for creating task-specific rules for evaluation, even more so for non-programmers. In regards to semantic evaluation, researchers have traditionally employed domain experts \[21\], while in this decade, crowdsourced validators \[17\], \[24\] are being used for semantic ontology validation. This paper proposes a customizable and scalable framework that evaluates syntactic and semantic aspects of ontology quality using SynEvaluator and SemValidator respectively.

- **SynEvaluator**: a tool that uses a rule-creation framework for allowing users to non-programmatically create rules during run-time and to set task-specific priorities for these rules.
A framework for syntactic and semantic quality evaluation of ontologies

- **SemValidator**: a tool that uses crowdsourcing for validation of semantic quality of enriched ontologies.

  In this paper, a Twitter-based expertise estimation algorithm is used to weight validators’ decisions.

  The source code for SynEvaluator and SemValidator is available on GitHub. They are also deployed on Heroku as web-applications.

  The rest of the paper is structured as follows: Section 2 describes related work in ontology quality evaluation. The proposed framework constituting of SynEvaluator and SemValidator is shown in section 3. Section 4 details the experiments done to show the efficacy and accuracy of the framework, through these tools. SynEvaluator is tested for its utility and accuracy for implementing syntactical quality evaluation rules by comparing it against a popular syntactic quality evaluation tool, OOPS! The efficacy of SemValidator is shown by conducting crowdsourced survey involving 28 validators on two popular ontologies, Stanford Pizza and Information Security ontology. Accuracy of TweetExpert algorithm on responses to both of these ontologies using multiple ML regression algorithms is shown. Finally, section 5 summarizes the contributions and suggests possible future directions of research.

2 Related Work

Ontology quality evaluation approaches can be broadly classified into a) syntactic and b) semantic quality evaluation approaches. Syntactic evaluation approaches primarily evaluate structural aspects of an ontology based on ontology development guidelines, common pitfalls, structural metrics etc. OntoClean, one of the earliest known works in this area, proposed a methodology for validating adequacy of relationships in an ontology based on notions drawn from philosophy such as essence, identity, and unity. Similarly, OntoQA proposed ontology evaluation on the basis of schema metrics and instance metrics. They stated that ‘goodness’ or ‘validity’ of an ontology varies between different users and domains. Gangemi et al. proposed structural, functional and usability-related measures using O2 and oQual, a meta-ontology and an ontology model for ontology evaluation and validation. Burton et al. proposed an ontology evaluation model based on semiotic theory. In order to apply the metrics proposed in these works, tools such as S-OntoEval drawn from semiotic theory and AktiveRank that ranked ontologies based on structural metrics like class match measure, density measure etc. were proposed. However, the tools proposed in these articles are either closed-source prototypes or theoretical frameworks and are not publicly available.

There are also a few publicly available closed-source ontology (syntactic) evaluation tools, such as OOPS!, DoORS and OntoMetrics. Ontology Pitfall Scanner (OOPS!) evaluates ontologies on the basis of established ontology quality rules related to human understanding, logical consistency, real world representation and modelling issues, and manually assigned priorities. DoORS, evaluates ontologies based on metrics drawn from Semiotic Theory while OntoMetrics uses metrics proposed in OntoQA. These tools, however are not flexible or customizable, and evaluate ontologies using a fixed set of rules or metrics. Users are unable to create/update customized rules and set task-specific priorities, which can be crucial as requirements and application scenarios vary. The available open source implementations require creation of new rules and priorities via programming which can be daunting for non-programmers.

Semantic evaluation approaches focus on semantic validity of concepts and relationships in an ontology. Traditionally, it has been formulated as a task requiring simple accept/reject decisions from

---

4 https://github.com/Remorax/SynEvaluator/
5 https://github.com/Remorax/SemValidator/
6 https://synevaluator.herokuapp.com/
7 http://semvalidator.herokuapp.com/
domain experts [2]. In the past few years, a good number of crowdsourced ontology evaluation approaches have been proposed. Hanika et al. [35] have developed the UComp Protégé plugin to provide a platform for crowdsourced workers to validate classes, subclasses, properties and instances. Kiptoo et al. [16] use crowdsourcing for axiom and assertion verification in ontologies as well as for verification of subclass-superclass relations by Amazon Mechanical Turks [24]. Pittet et al. used crowdsourced workers to propose changes related to addition, deletion and substitution errors [25]. Zhang et al. [37] used crowdsourced workers to obtain written feedback (comments/suggestions/references) for making final validation decisions. Requiring complex tasks (such as making data quality decisions or requiring written feedback) from crowdsourced workers can be expensive and unscalable as the size of ontology and/or number of workers increases. Some approaches [35], [17], [25] used quality control mechanisms like majority voting that are debatable. Noy et al. [24] addressed this by using qualification questions and spam filtering techniques. While these mechanisms eliminate spammers, it may not be applicable where large number of workers have some degree of knowledge but only few are experts. Therefore, an assessment of domain expertise on a continuous scale would be useful as a quality control metric. Further, an integrated quality evaluation framework that seeks to evaluate ontologies on both syntactic and semantic aspects, and also addresses the above-mentioned problems would be useful for a holistic and integrated approach to quality evaluation of enriched ontologies.

3 Proposed Framework

In this paper, an ontology evaluation framework that combines automated syntactic evaluation and human-centric semantic evaluation is proposed. SynEvaluator aims at increasing user flexibility by allowing customized rule creation at runtime, as well as scalability (with respect to user base) by proposing an approach that removes the need for programming. SemValidator proposes a crowdsourcing based approach that uses the validators’ Twitter profiles for quality control.

The framework’s work flow is shown in Figure 3. The input to quality evaluation process is an enriched ontology. The ontology may have been enriched with concepts, relations and instances using some automated and semi-automated algorithms. The ontology may contain syntactic quality errors (due to violation of ontology development guidelines) and/or semantic errors (due to wrongly enriched concepts). In the figure, for clarity, concepts and relationships with potential syntactic violations are bordered in yellow, and those that do not contain such violations are outlined in blue. Also, concepts that potentially contain semantic violations are highlighted in yellow colour and concepts that are semantically valid are highlighted in green.

An ontology engineer uploads an enriched ontology to SynEvaluator, and creates syntactic quality evaluation rules. The rules created using a theoretical rule creation framework are applied on the parsed ontology object through SynEvaluator’s ontology evaluation module. This returns a list of detected violations and the elements causing these violations. Using these elements as suggestions, the ontology engineer can fix violations in an iterative manner. The iterations may be repeated as needed. Then, the ontology engineer can provide this ontology as input to SemValidator so that it can be validated for semantic quality. As part of semantic validation, the ontology is provided to crowdsourced validators who give their accept or reject validation decisions for each of the enriched concepts, relations and instances. Simultaneously, an estimate of the domain knowledge of each of these validators is calculated from their Twitter profile using the TweetExpert algorithm. These scores alluded to as ‘TweetExpert scores’, are used as a quality control mechanism to ensure that the decisions of crowdsourced validators are given weightages according to their knowledge of the ontology domain. Finally, the output of this algorithm is
used to take the final accept/reject decision for each enriched element, resulting in an ontology with both good syntactic and semantic quality.
3.1 Stage 1: SynEvaluator

In this section, the underlying terminology used in SynEvaluator and rule creation framework is illustrated with examples. Further, the implementation of SynEvaluator as a web application is detailed. Finally, the section ends with a discussion on potential benefits and limitations of SynEvaluator.

Defining the Rule Creation Framework  SynEvaluator allows users to create rules at run-time. The rules are constructed from individual components like “Subjects”, “Clauses” or “Operator Expressions”. The operational definitions of these and other relevant terms are:

- **Ontological Element**: Refers to any element that forms a constituent part of an ontology. It refers to any primary component (classes, individuals, properties), their related elements (subclasses, domains, ranges), annotations (labels, descriptions, comments) or attributes (ID, language, namespace).
- **Rule**: Refers to a sequence of one or more clauses, optionally connected by one or more operator expressions that returns either one or more ontological elements or a boolean value.
- **Clause**: Refers to a transformation applied on an ontological element(s) to return either one or more ontological elements or a boolean value.
- **Operator Expression**: An expression used to compare and/or connect non-empty sequences of clauses to return a boolean value.
- **Subject**: Refers to an ontological element (typically primary components such as classes, individuals and properties), that is subjected to transformations carried out through sequential clauses to form a rule.

![Fig. 4: Structure of supported expressions](image)

Using these concepts, the expressions supported by SynEvaluator are formally defined as shown in Figure 4. The notation used is similar to *RegEx* notation with ∗ denoting zero to infinity, + denoting one to infinity, and ? denoting zero or one occurrences. A ‘Subject’ comprises the beginning and is always the first keyword in a rule in out proposed framework. It goes through a series of transformations as defined by sequences of clauses. Clauses can be of of two types: a) Extractive Clauses and b) Functional Clauses. Extractive Clauses consist of (Predicate, Object) pairs that use the Predicate to execute a transformation on the return value from the previous clause using the Object as argument. More specifically, object specifies the type of element ‘extracted’ by the predicate, and elements satisfying this (Predicate, Object) pair are returned as output. Functional clauses, on the other hand, consist of (Predicate, Function) pairs that involve executing a function of type described by predicate on the return value from the previous clause. These clauses typically check for existence of a certain functional property and thus return a boolean value in response.

Currently, two kinds of functional properties are supported: i) ontological (or structural) properties, that execute ontology-level functions (such as uniqueness, validity, consistency etc) on ontological elements, and ii) linguistic properties that linguistically analyze text (such as checking for polysemes, conjunctions etc) returned from previous clauses. In case of ‘False’ value returned by a Functional Clause or an empty list (no
Extractive Clauses can also be chained together to form clause sequences. Since functional clauses return a boolean value, they cannot be chained further. Clause sequences can be compared through Operator Expressions. Operator Expressions essentially consist of a ‘Predicate’ indicating operator type, followed by an ‘Operator’ keyword. Operator Expressions comprise of two main categories of operators, namely: (a) Logical Operators and (b) Comparative Operators. Logical Operators like ‘And’, ‘Or’ and ‘Not’ are used to create logical combinations of clause sequences. Comparative Operators like ‘Equality’, ‘Inverse’ and ‘Synonymy’ are used to compare return values.

| Subject             | Extractive Clause | Functional Clause | Operator Expression |
|---------------------|-------------------|-------------------|---------------------|
| Has Related Element | Has Ontological Property | Uses Comparative Operator |
| Has Attribute       | Has Linguistic Property | Uses Logical Operator |
| Domain              | ID Consistency     | Equality          |
| Subclass            | Uniqueness         | Inverse           |
| Has Related Element | Domain             | ID                |
| Disjoint Class      | Text Validity      | And               |
| Has Attribute       | Uniqueness         | Not               |
| ID                  | Contains Polysemes | Or                |
| Language            | Contains Conjunctions |                |

Table 1 summarizes keywords supported by the proposed framework. Note that the table lists 8 Subject keywords, while for Extractive Clauses, Functional clauses and operator expressions it lists 2 predicates and 5 objects, functions and operators respectively. This is due to differing syntax followed by each expression type. Also, every predicate has a list of valid Objects/Functions/Operators. This is shown in the table through bracketed numbering. For example, the valid Predicates for ‘Has Attribute’ are ‘ID’ and ‘Language’. The complete list of supported keywords is provided over here.

Examples of Created Rules Figure 5 shows examples of 2 rules. In both examples 1 and 2, ‘Property’ is the ‘Subject’ of the rule. ‘hasRelatedElement Domain’, ‘hasRelatedElement Range’ and ‘hasOntologicalProperty Uniqueness’ are all clauses. In rule 1, the ‘hasRelatedElement Domain’ clause carries out a transformation that uses the ‘hasRelatedElement’ predicate to extract ‘Domain’ elements. A similar clause is used for extracting ‘Range’ elements. These two clauses (or clause sequences of length are combined using the operator expression ‘usesLogicalOperator And’. This rule thus necessitates non-null values for both Domain and Range elements for each element of ‘Subject’, in this case, ‘Property’. Properties that do not contain both elements are therefore returned as ontological elements containing violations.

Example 2 shows a rule where clauses have been chained together to constitute a clause sequence of length 2. This sequence consists of the extractive clause ‘hasRelatedElement Domain’ followed by functional clause ‘hasOntologicalProperty Uniqueness’. The extractive clause essentially extracts Domain element(s) of each Property. Then, the functional clause applies ‘Uniqueness’ function on ontological

https://bit.ly/3zfdI8f
Fig. 5: Examples of rules implemented by SynEvaluator

elements returned by previous clause with function type defined by Predicate “hasOntologicalProperty”. In case of non-existence of domain for a particular property or existence of multiple domains, this rule would return that property as containing a violation due to violation in first and second clauses respectively.

Fig. 6: The SynEvaluator interface. Subjects are highlighted in blue, Clauses in yellow, Operator Expressions in green and Rule Priorities in red. Among Clauses, Extractive Clauses are shown in solid lines while Functional Clauses are outlined with dotted lines.

Proposing SynEvaluator: the final web application The rule creation framework proposed above is used to create a web application for use by the ontology engineer. The primary interface of this application, SynEvaluator, is shown in Figure 6. It allows the users (ontology engineers) to use dropdown menus to create functional and extractive clauses, operator expressions and thus, rules, as well as set priorities for these rules. Users can choose between ‘Low’, ‘Medium’ and ‘High’ priorities based on the task-specific importance of the rule. Figure 8 shows, with the help of an activity diagram, how a user could create an appropriate rule using SynEvaluator. Finally, after uploading the enriched ontology and creating the rules, SynEvaluator parses the ontology using its parsing module and executes the created rules (Figure 3). Post evaluation, the user is presented with a list of violating elements along with count and priority of each valid rule.
Benefits and Limitations  The proposed framework makes it significantly easier for non-programmers to create customised rules dynamically. Also, compared to previous quality evaluation tools, due to the framework’s ability to reuse keywords to create new rules, the developer effort required to hard-code rules is minimised. Another major benefit is that the proposed framework can potentially be used to query over entire OWL language. This can be done as any ontological element/attribute can be extracted using extractive clauses and the appropriate function executed on them. Lastly, due to functional clauses, it is possible to execute ontological or ontology-level functions like normal query languages and use linguistic analysis on text. This is particularly useful in quality evaluation while applying appropriateness checks on IDs or labels.

Example 1: Element hasDescriptiveElement Comment usesLogicalOperator Or hasDescriptiveElement Description

Example 2: Class hasAttribute ID usesComparativeOperator Dissimilarity hasRelatedElement EquivalentClass hasAttribute ID

Example 3: Element usesLogicalOperator Not hasDescriptiveElement Label hasLinguisticProperty ContainsConjunctions

Example 4: Element usesLogicalOperator Not hasDescriptiveElement Label hasLinguisticProperty ContainsPolysemes

In Figure 7 (Example 1), it can be seen that properties with missing domain or range, a common quality violation, could be detected through a combination of logical operators and extractive clauses. Another common violation is related to the absence of annotations (comments, descriptions, etc.) for an ontological element. This can be converted into a rule, once again through Extractive Clauses as shown in Figure 7 (Example 1). A third quality violation would be when similar (or synonymous) classes are incorrectly defined as equivalent classes. This can be defined as in Figure 7 (Example 2), where the comparative operator ‘Dissimilarity’ is used to test for semantic similarity through cosine similarity of label embeddings. As mentioned previously, it is also possible to perform linguistic tests unlike other query languages. A basic example of this is detection of conjunctions in a label as shown in Figure 7 (Example 3). This is useful to identify quality violations where different concepts are merged in the same class using conjunctions. A more advanced example of a linguistic test would be a rule detecting polysemous elements (Figure 7 Example 4). This is useful in detecting violating elements that have labels denoting differing conceptual ideas in differing senses. SynEvaluator implements this check through the use of WordNet’s synsets to find out how many senses a word can have. Once these rules are created, the ontology engineer can add domain/range elements; annotations; remove synonymous equivalent classes and fix classes with conjunctions and polysemes as appropriate. SynEvaluator can thus help in fixing structural and linguistic quality violations.

The current version of SynEvaluator has a few limitations. It is currently only possible to chain clauses together or use operators to compare chained clauses. As a result, it is not possible to create rules with multiple lines. One major consequence of this is that variable assignment is not supported, and it is not possible to create a variable in one line and refer to it in another, as part of the same rule. Aggregation
operators, such as ‘Count’ or ‘Sum’, are currently not supported either. Finally, it is not possible to create rules that require reasoning. The described limitations shall be addressed in future iterations of SynEvaluator. In spite of the limitations, the current framework (as shown in Experiments section) is still able to support creation of the majority of quality evaluation rules.

![Activity Diagram for creating a rule in SynEvaluator](image)

Fig. 8: Activity Diagram for creating a rule in SynEvaluator

### 3.2 Stage 2: SemValidator

SemValidator uses a crowdsourced approach to semantically evaluate ontologies. The key feature of SemValidator is that it does not require ontology engineers, domain experts or knowledge of OWL language for validation. This is useful in a crowdsourced setting, where validators may have varying degrees of expertise and knowledge. If further necessitates the use of appropriate quality control mechanisms. To ensure quality, SemValidator uses TweetExpert algorithm to calculate expertise score of a crowdsourced validator, which is then used to weigh their decisions. This section describes the approach used by TweetExpert and justifies the choices made. This is then followed by a discussion on the assumptions made by TweetExpert and the feasibility of the assumptions in the context of crowdsourcing. The section finally ends with a description of the implementation of SemValidator and how it can be used by crowdsourced validators.

**TweetExpert algorithm** TweetExpert algorithm takes Twitter usernames of validators as input. For each of the validators, it calculates two scores: a) a ‘TweetSim’ score and b) a ‘FriendSim’ score. A ‘TweetSim’ score is intended to assess the similarity of the validator’s tweets to the domain of the ontology while ‘FriendSim’ estimates the domain similarity of the validator’s friends (pages they are following). To calculate the ‘TweetSim’ score, the validator’s ‘n’ most recent tweets are extracted from their profile and their semantic similarity with the domain keyword is computed using the Universal Sentence Encoder.
Here the ‘domain keyword’ is manually chosen as the word most relevant to the domain of the ontology. For instance, “Pizza” for Stanford Pizza ontology [8] and “Information Security” for the ISO 27001 based “Information Security” ontology [10]. The current implementation uses only one keyword, but it is possible to compute similarity scores with multiple keywords and average these scores for greater accuracy. The similarities are sorted in decreasing order. Out of n similarities, the top-K similarities are chosen. These top-K similarities are then averaged to yield ‘TweetSim’ score. A similar approach is used to calculate ‘FriendSim’ score. The ‘m’ most recent friends are extracted to calculate their ‘TweetSim’ scores. After sorting in decreasing order, the top-K most similar scores are averaged to yield ‘FriendSim’ score. The reason behind extracting the most recent tweets and friends is to get a better estimate of the validator’s current knowledge and interests. On the other hand, a top-K average helps in both filtering out occasional out-of-domain tweets from domain experts and smoothening out the effects of coincidental in-domain outliers from non-experts. The value of K is thus appropriately empirically chosen such that it is large enough to not include in-domain outliers from laymen, but small enough to exclude any out-of-domain tweets from experts. The pseudocode for TweetExpert is shown in Algorithm 1.

**Algorithm 1: TweetExpert algorithm**

**Result:** TweetExpert score

**Function** calculate_tweet_similarity(username):
    user_tweets := extract_tweets(username)
    tweet_similarities := calculate_USE_similarity(user_tweets, domain_name)
    best_tweet_similarities := get_top_K_tweets(tweet_similarities)
    tweet_similarity_score = average(best_tweet_similarities)
    return tweet_similarity_score

**Function** calculate_friend_similarity(username):
    user_friends := extract_friends(username)
    user_friends_scores := foreach friend ∈ user_friends do calculate_tweet_similarity(friend)
    best_friend_scores := get_top_K_friends(user_friends_scores)
    friend_similarity_score = average(best_friend_scores)
    return friend_similarity_score

tweet_sim := calculate_tweet_relevance(username)
friend_sim := calculate_friend_relevance(username)
score := ML_predict_score(tweet_sim, friend_sim)

Finally, the calculated ‘TweetSim’ and ‘FriendSim’ scores are input to a pre-trained Machine Learning regression algorithm that predicts the final TweetExpert score using the two scores as feature vectors. The current implementation uses Epsilon-Support Vector Regression (SVR), since it was experimentally found to yield best results (shown in Experiments section). However, the system uses strategy software design pattern [29] which enables easy interchange of regression algorithms. The TweetExpert score is calculated for each of the validators by repeating the process described above. The final decision is taken using a weighted majority voting algorithm with the TweetExpert scores being used as weights. The TweetExpert scores provide a way to estimate a confidence value for the decisions input by each validator. This is
particularly crucial as a quality control metric in non-probabilistic sampling techniques like crowdsourcing where number of validators can grow exponentially in count and diversity.

**Assumptions** SemValidator makes reasonably grounded assumptions to establish efficacy and suitability in a crowdsourced setting. For example, about 49% of crowdsourced workers whose primary source of income is Amazon Mechanical Turk, a popular crowdsourcing platform, are in the 18-29 age range. As of February 2021, about 42% of Americans in the 18-29 age range use Twitter, with this age group being the most active demographic on Twitter. It is also assumed that in order for expertise estimation to work, Twitter users tweet reasonably frequently. The average number of tweets per day per user, according to a 2016 study, is 4,422, which translates to over 1600 tweets a year. This is typically sufficient since $K \sim 50 – 100$. Finally, SemValidator assumes workers have public Twitter profiles, which would enable tweets and friends of a worker to be extracted. This assumption is predicated upon a recent 2019 survey that showed 87% of Twitter users in USA have public accounts. Also, as part of future work, SemValidator would allow login through Facebook and Linkedin as well and a similar algorithm for expertise estimation could be used. This is expected to further increase the applicability of this expertise-based approach for crowdsourcing.

Fig. 9: The main validation interface of SemValidator. The Stanford Pizza ontology has been uploaded on SemValidator, with its concepts in blue and the enriched concepts in green. One of the enriched concepts, “Tandoori Pizza” has been selected, with options to “Accept” and “Reject” it.

---

9 https://pewrsr.ch/3vX5q2G
10 https://bit.ly/3z5IDn9
11 https://bit.ly/3fSjWmA
12 https://bit.ly/3pmMc3O
Proposing SemValidator: the final validation workbench  The proposed workflow is used to develop a validation workbench for crowdsourced workers that allows for accepting or rejecting enriched concepts, relations and instances in an enriched ontology. The main validation interface of this workbench, called SemValidator, is shown in Figure 9. This application integrates Twitter authentication and uses the TweetExpert algorithm for calculating validator expertise. SemValidator allows for two types of users: (a) the administrator and (b) the validator. The administrator, typically the ontology engineer, can upload/delete ontologies to be validated and also access decisions made by validators. When the validator selects an ontology to validate, the ontology is served using the WebVOWL ontology visualization software. The enriched concepts and instances are highlighted in green, and on selection, enable the validators to select accept/reject decisions accordingly. Enriched relations, also in green, may also be accepted/rejected independent of the concepts they relate. The validators’ decisions are recorded by SemValidator and logged to a database. The administrator can download this database after the crowdsourcing survey is completed, evaluate validator expertise using their Twitter usernames and then make final accept/reject decisions.

The syntactic and semantic evaluation aspects of the framework may be used independently of each other. However, utilizing the framework in an integrated manner as shown in Figure 3 is expected to give best results. This way, syntactical violations can be detected easily and in an automated and customizable manner, while semantic violations can now be detected more accurately by crowdsourced validators. The resulting ontology has enhanced syntactic and semantic ontological quality and is now fit for reuse.

4 Experiments

Stanford Pizza and ISO-IEC 27001 Information Security ontologies are evaluated using the proposed framework to demonstrate ontology quality evaluation. Since the focus of this work is on enriched ontologies, these are manually enriched with concepts, properties and instances before quality evaluation. The RDF triples used to enrich Pizza and Information Security ontologies are provided over here 13. Only 5 triples were chosen for this iteration, considering that ontology enrichment is as an iterative process and the count of triples per iteration is expected to be of this order. For Pizza, the domain-specific webpages used for extraction consisted of culinary articles 14 and food travel blogs 15. For Information Security they consisted of informative articles and product pages by Cisco 16, Barracuda 17 and SearchSecurity 18. Please note that some relevant data in this section is shown through external links due to space constraints.

4.1 Syntactic Quality Evaluation using SynEvaluator

This section attempts to evaluate the applicability and accuracy of SynEvaluator in creating rules and detecting violations respectively. It is hypothesized that SynEvaluator’s reusable theoretical framework for priority-specific rule creation increases its applicability for diverse range of tasks, but without compromising on violation-detection accuracy. To prove these claims, SynEvaluator is compared against OOPS! 27, a popular, publicly available SOTA tool that allows qualitative rule evaluation using rules. Moreover, OOPS! compiles 41 most commonly observed pitfalls drawn from several popular works on

13 https://bit.ly/3z5lGR1
14 https://bit.ly/3yYGGZP
15 https://bit.ly/3iiEtCm
16 https://bit.ly/3wXRHZj
17 https://bit.ly/34P1UN9
18 https://bit.ly/3wXRJQV
ontology quality evaluation [12], [23], [34] and the ones described in the OOPS! catalogue [26] are chosen as rules for implementation. This allows appropriate assessment of applicability for rule creation, while also allowing accuracy evaluation by comparing SynEvaluator’s detected violations to that of OOPS! Note that while SynEvaluator’s suitability for rule creation is being assessed here by comparing with rule-based evaluation approaches, it cannot be compared yet with other contemporary works focusing on metric-based evaluation. This is being planned as part of future work by introducing metric-to-rule conversion, which would allow metrics to be framed as rules. One possible way this could be done is by adding conditions, such as comparison operators, after metrics to form rules.

Table 2: No of pitfalls implemented by SynEvaluator (SE) Vs OOPS!

|                    | SE Implemented | OOPS! Implemented | SE Not implemented | OOPS! Not implemented |
|--------------------|----------------|-------------------|--------------------|-----------------------|
| Implemented        | 29             | 33                | 4                  | 7                     |
| Not implemented    | 1              | 0                 | 7                  | 0                     |

SynEvaluator’s framework can be used to successfully formulate 30 of the 41 pitfalls compiled in the catalogue, compared to OOPS! which can automate 33. As shown in Table 2, the majority of pitfalls (29 of 41) can be successfully implemented by both OOPS! and SynEvaluator. 7 cannot be implemented by either of them, and this involves rules that require human knowledge and reasoning (such as overspecialization of ontology hierarchy, usage of wrong relations, etc.). 1 of the pitfalls (involving linguistic polysemy detection) could be implemented by SynEvaluator but not by OOPS!. There are 4 pitfalls implemented by OOPS! which SynEvaluator cannot check. This includes rules that check for undeclared disjoint concepts, undeclared transitive properties, equivalent classes, etc. Supporting such rules involves a higher degree of ontological reasoning that SynEvaluator is incapable of. Nevertheless, SynEvaluator’s ability to implement vast majority of quality evaluation rules off-the-shelf in a customizable manner increases confidence in its applicability for future evaluation tasks that may involve more rules.

For comparing accuracy of violation detection, SynEvaluator’s reported violations are compared with OOPS! [27]. Given the absence of ground truth and the widespread popularity of OOPS!, it is assumed that OOPS! uses valid, non-errorneous rule checks for the implemented pitfalls. It can be observed that the violations reported by SynEvaluator match those of OOPS! in all pitfalls except for P22. OOPS! mentions that these could be due to inconsistent naming conventions, however SynEvaluator is unable to detect any such errors in the ontologies. This may be a result of the incorrect naming checks currently used. But for the other pitfalls, both tools returning the same number of violations suggests that for the implemented framework, SynEvaluator is accurate in evaluating rules and detecting pitfalls.

4.2 Semantic Quality Validation using SemValidator

SemValidator uses a crowdsourced survey to semantically validate the enriched Pizza and Information Security ontologies. Crowdsourced validators were invited by tweeting a description of the task along with the link to the SemValidator application. Among the 31 validators that participated in the survey, 3 had private Twitter accounts so their responses were discarded. The 28 validators with public accounts included 5 validators with a background in Cybersecurity, 4 in Artificial Intelligence, 5 in Software Engineering, 4 in Healthcare, 3 in Food domain (such as chefs and culinary experts), 3 in Literature, 2 in Politics and 1 each in Fashion and Pure Science. These diverse bunch of validators were asked to self-identify their domain of
expertise for statistical purposes before participating in the survey. To preserve validator anonymity, the anonymized results of the survey are provided here. After completion of the survey for both ontologies, ‘TweetSim’ and ‘FriendSim’ scores were calculated for all 28 validators using the TweetExpert algorithm, described previously. TweetExpert took 15 seconds to execute for our values of $K = 20$ and $K' = 5$ per validator profile. Finally, TweetExpert score was calculated by experimenting with 4 standard regression algorithms: Linear Regression, Random Forest Regression, K-Nearest Neighbours (KNN) Regression and Epsilon-Support Vector Regression (SVR). A standard majority voting where all decisions have equal weightage is considered as a naive baseline.

The input features for each algorithm were ‘TweetSim’ and ‘FriendSim’ scores of a validator and the label was the ratio of correct answers (to total answers) given by them. This was done to ensure validators with more correct answers received higher ‘TweetExpert’ scores using similarity scores as feature vectors. Since a gold standard for the correct answers did not exist, CISOs were asked to manually validate the enriched triples from the information security domain, while the authors themselves validated the triples from the pizza domain. To determine the best among the 4 regression algorithms for score prediction, 7-fold cross validation was carried out. The 28 responses (now reduced to feature vectors and labels) available for both ontologies were divided into 7 folds of 4 responses each. 5 folds were used for training, 1 for validation and 1 for testing. For each chosen test fold, immediately succeeding fold was chosen as the validation set and all the other folds were used for training.

Different sets of hyperparameters were considered for each regression algorithm. The best set of hyperparameters for each algorithm was determined by selecting the algorithm with highest accuracy on the validation set, calculated by determining the proportion of correctly answered questions to the total number of questions. The predicted answer to a question is determined by using weighted majority voting algorithm on the predicted expertise scores in the validation set. The higher the number of correct answers, the closer the predicted scores are to their actual value. The same procedure was followed for calculating the accuracy on the test folds. Accuracy obtained on test and validation sets for each of these algorithms is shown in Table 3 for both ontologies. Epsilon Support Vector Regression was found to yield best results on both datasets.

To determine whether training score prediction algorithms requires feature vectors from the same domain, an SVR model is trained on a more generic dataset, i.e the Pizza dataset and tested on Information Security, a niche dataset. The model hyperparameters are chosen as the ones that performed best on the Information Security dataset earlier. Varying percentages of the Pizza dataset are used to construct training subsets, to observe variation in test accuracy with subset size. To ensure randomness while choosing these subsets, 10 experimental trials were conducted where the dataset is shuffled before each trial and trained on the first $x\%$ of samples, where $x$ is the percentage of the training dataset chosen. The results are shown in Table 4. The accuracy monotonically increases with increase in size of training subset. The results suggest that a model pre-trained to predict the expertise score on one domain could be reused multiple times in different domains, even if they are niche.

Table 3 shows that the best-performing algorithm (TweetExpert with SVR followed by weighted majority voting) significantly outperforms naive majority voting. It can be observed that naive majority voting gives particularly bad results for Information Security, a relatively niche domain, than Pizza, a domain known more to laymen as well. When replacing naive majority voting with TweetExpert + SVR, a drastic increase in performance in Information Security (55.55%) vs Pizza (20%) can be observed. Even the worst performing regression algorithm gives an increase of 38.8% and 14% respectively. As a result, it can be inferred that estimating expertise using TweetExpert can be particularly useful for quality control in niche domains.

---

19 https://bit.ly/3pvzbFh
20 https://bit.ly/3wUa9SA
Table 3: Mean Validation and Test Accuracy scores on Pizza and Security datasets

| Algorithm       | Pizza   | Security | Pizza   | Security |
|-----------------|---------|----------|---------|----------|
|                 | Val     | Test     | Val     | Test     |
| Majority Voting | 71.43   | 71.43    | 51.43   | 51.43    |
| Linear Regression| 85.71  | 83.57    | 68.57   | 71.43    |
| Random Forest   | 80.0    | 81.43    | 68.57   | 74.28    |
| KNN             | 85.71   | 80.0     | 77.14   | 77.14    |
| SVR             | 88.57   | 85.71    | 80.0    | 80.0     |

Table 4: Mean accuracy scores on test dataset (Information Security) with variation in % of training dataset (Pizza).

| Dataset% | 10% | 25% | 50% | 75% | 100% |
|----------|-----|-----|-----|-----|------|
| Accuracy | 70% | 76% | 86% | 92% | 100% |

5 Conclusion and Future Work

Ontology evaluation is a critical stage of any ontology engineering process. In this paper, SynEvaluator and SemValidator are proposed for syntactic and semantic evaluation of an enriched ontology respectively. Although the work focuses on an enriched ontology, it can be easily extended for evaluation of any generic ontology.

SynEvaluator automatically evaluates ontologies using customised rules created dynamically at runtime, and improves on previous quality evaluation approaches in a variety of ways. Firstly, it offers greater flexibility to the user in terms of creating, updating and deleting custom rules as well as setting priorities. Secondly, it proposes rule creation by the usage of a novel theoretical framework that factors rules into sequences of ‘clauses’ and ‘operator expressions’. This facilitates creation of an interactive interface that makes it easier for non-programmers to dynamically create rules. In addition, chaining together independent keywords can facilitate creation of a large number of rules without requiring additional developer programming. SemValidator improves on previously proposed crowdsourced ontology validation approaches by incorporating a Twitter-based quality control mechanism. The TweetExpert algorithm is proposed for calculating the expertise score of a validator using the tweets and friends extracted from their Twitter profile as input.

The efficacy of SynEvaluator was shown by implementing rules to detect pitfalls and the accuracy of the detected violations was compared with publicly available OOPS! tool. Semantic quality evaluation using SemValidator is performed on both Pizza and Information security ontologies with the help of a crowdsourced survey of 28 validators. The experimental results showed a significantly higher than naive majority voting.

The experimental results are encouraging but also can be used as an aid to further extend the research work. For example, SynEvaluator can be expanded to support additional, more complex operations such as using parentheses, arithmetic operators, variable assignment etc. Although the TweetExpert algorithm used by SemValidator currently calculates expertise using only tweets and friends as feature vectors, it could be extended to use additional information such as Twitter lists, followers, tweet metadata, etc.
References

1. Alani, H., Brewster, C., Shadbolt, N.: Ranking ontologies with AKTiveRank. In: International Semantic Web Conference. pp. 1–15. Springer (2006)
2. Amardeilh, F., Laublet, P., Minel, J.L.: Document Annotation and Ontology Population from Linguistic Extractions. In: Proceedings of the 3rd International Conference on Knowledge Capture. pp. 161–168 (2005)
3. Beckett, D., Berners-Lee, T., Prud’hommeaux, E., Carothers, G.: Rdf 1.1 turtle. World Wide Web Consortium pp. 18–31 (2014)
4. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. Scientific American 284(5), 34–43 (2001)
5. Burton-Jones, A., Storey, V.C., Sugumaran, V., Ahluwalia, P.: A semiotic metrics suite for assessing the quality of ontologies. Data & Knowledge Engineering 55(1), 84–102 (2005)
6. Cer, D., Yang, Y., Kong, S.y., Hua, N., Limtiaco, N., John, R.S., Constant, N., Guajardo-Céspedes, M., Yuan, S., Tar, C., et al.: Universal Sentence Encoder. arXiv preprint arXiv:1803.11175 (2018)
7. Dividino, R.Q., Romanelli, M., Sonntag, D., et al.: Semiotic-based Ontology Evaluation Tool (S-OntoEval). In: LREC (2008)
8. Drummond, N.: Stanford pizza ontology. https://protege.stanford.edu/ontologies/pizza/pizza.owl, accessed: 2021-03-28
9. Duque-Ramos, A., Fernández-Breis, J.T., Iniesta, M., Dumontier, M., Aranguren, M.E., Schulz, S., Aussenac-Gilles, N., Stevens, R.: Evaluation of the OQuaRE Framework for Ontology Quality. In: Expert Systems with Applications. vol. 40, pp. 2696–2703. Elsevier (2013)
10. Fenz, S., Goluch, G., Ekelhart, A., Riedl, B., Weippl, E.: Information security fortification by ontological mapping of the iso/iec 27001 standard. In: 13th Pacific Rim International Symposium on Dependable Computing (PRDC 2007). pp. 381–388. IEEE (2007)
11. Gangemi, A., Catenacci, C., Ciaramita, M., Lehmann, J.: A theoretical framework for ontology evaluation and validation. In: SWAP. vol. 166, p. 16 (2005)
12. Gómez-Pérez, A.: Evaluation of taxonomic knowledge in ontologies and knowledge bases (1999)
13. Guarino, N., Welty, C.: Evaluating ontological decisions with ontoclean. Communications of the ACM 45(2), 61–65 (2002)
14. Hearst, M.A.: Automatic Acquisition of Hyponyms from Large Text Corpora. In: Proceedings of the 14th conference on Computational linguistics. vol. 2, pp. 539–545. Association for Computational Linguistics (1992)
15. Iyer, V., Mohan, L., Bhatia, M., Reddy, Y.R.: A survey on ontology enrichment from text. In: Proceedings of the 16th International Conference on Natural Language Processing. pp. 95–104. NLP Association of India, International Institute of Information Technology, Hyderabad, India (Dec 2019), https://aclanthology.org/2019.icon-1.22
16. Kiptoo, C.C.: Ontology enhancement using crowdsourcing: a conceptual architecture. International Journal of Crowd Science (2020)
17. Kontokostas, D., Zaveiri, A., Auer, S., Lehmann, J.: Triplecheckmate: A tool for crowdsourcing the quality assessment of linked data. In: International Conference on Knowledge Engineering and the Semantic Web. pp. 265–272. Springer (2013)
18. Lantow, B.: OntoMetrics: Putting Metrics into Use for Ontology Evaluation. In: KEOD. pp. 186–191 (2016)
19. Lozano-Tello, A., Gómez-Pérez, A., Ontometric: A Method to Choose the Appropriate Ontology. In: Journal of Database Management. vol. 2, pp. 1–18. IDEA Group Publishing (2004)
20. Maedche, A., Staab, S.: Ontology Learning for the Semantic Web. Intelligent Systems 16(2), 72–79 (2001)
21. Makki, J., Alquier, A.M., Prince, V.: An nlp-based ontology population for a risk management generic structure. In: Proceedings of the 5th international conference on Soft computing as transdisciplinary science and technology. pp. 350–355 (2008)
22. McDaniel, M., Storey, V.C., Sugumaran, V.: Assessing the Quality of Domain Ontologies: Metrics and an Automated Ranking System. Data & Knowledge Engineering 115, 32–47 (2018)
23. Noy, N.F., McGuinness, D.L., et al.: Ontology development 101: A guide to creating your first ontology (2001)
24. Noy, N.F., Mortensen, J., Musen, M.A., Alexander, P.R.: Mechanical turk as an ontology engineer? using microtasks as a component of an ontology-engineering workflow. In: Proceedings of the 5th Annual ACM Web Science Conference. pp. 262–271 (2013)
25. Pittet, P., Barthélémy, J.: Exploiting users’ feedbacks: Towards a task-based evaluation of application ontologies throughout their lifecycle. In: International conference on knowledge engineering and ontology development. vol. 2 (2015)
26. Poveda, M.: Catalogue of common pitfalls. [http://oops.linkeddata.es/catalogue.jsp](http://oops.linkeddata.es/catalogue.jsp) accessed: 2021-03-28
27. Poveda-Villalón, M., Gómez-Pérez, A., Suárez-Figueroa, M.C.: Oops!: A pitfall-based system for ontology diagnosis. In: Innovations, Developments, and Applications of Semantic Web and Information Systems, pp. 120–148. IGI Global (2018)
28. Poveda-Villalón, M., Suárez-Figueroa, M.C., Gómez-Pérez, A.: Validating Ontologies with OOPS! In: International Conference on Knowledge Engineering and Knowledge Management. pp. 267–281. Springer (2012)
29. Richard, E., RALPH, H., JOHNSON, R., et al.: Design patterns: Elements of reusable object-oriented software (1995)
30. Sanagavarapu, L.M., Iyer, V., Reddy, Y.R.: OntoEnricher: A Deep Learning Approach for Ontology Enrichment from Unstructured Text. arXiv preprint arXiv:2102.04081 (2021)
31. Schober, D., Tudose, I., Svatek, V., Boeker, M.: Ontocheck: verifying ontology naming conventions and metadata completeness in protégé 4. In: Journal of biomedical semantics. vol. 3, pp. 1–10. BioMed Central (2012)
32. Tartir, S., Arpinar, I.B., Sheth, A.P.: Ontological Evaluation and Validation. In: Theory and applications of ontology: Computer applications, pp. 115–130. Springer (2010)
33. Tibaut, A.: Ontology Evaluation. [https://github.com/atibaut/ontology-evaluation](https://github.com/atibaut/ontology-evaluation) (Online; accessed on 15-Mar-2021)
34. Vrandečić, D.: Ontology Evaluation. In: Handbook on Ontologies, pp. 293–313. Springer (2009)
35. Wohlgenannt, G., Sabou, M., Hanika, F.: Crowd-based ontology engineering with the ucomp protégé plugin. Semantic Web 7(4), 379–398 (2016)
36. Wong, W., Liu, W., Bennamoun, M.: Ontology Learning from Text: A Look Back and into the Future. ACM Computing Surveys (CSUR) 44(4), 1–36 (2012)
37. Zhang, Y., Saberi, M., Chang, E.: Semantic-based lightweight ontology learning framework: a case study of intrusion detection ontology. In: Proceedings of the international conference on web intelligence. pp. 1171–1177 (2017)