Literally Better: Analyzing and Improving the Quality of Literals

Wouter Beek [a] Filip Ilievski [a] Jeremy Debattista [b] Stefan Schlobach [a]
[a] *VU University Amsterdam*
e-mail: {w.g.j.beek,k.s.schlobach}@vu.nl
[b] *University of Bonn & Fraunhofer IAIS*
e-mail: debattis@iai.uni-bonn.de

AbstractQuality is a complicated and multifarious topic in contemporary Linked Data research. The aspect of literal quality in particular has not yet be rigorously studied. Nevertheless, analyzing and improving the quality of literals is important since literals form a substantial (one in seven statements) and crucial part of the Semantic Web. Specifically, literals allow infinite value spaces to be expressed and they provide the linguistic entry point to the LOD Cloud. We provide a toolchain that builds on the LOD Laundromat data cleaning and republishing infrastructure. This toolchain alows us to analyze the quality of literals on a very large scale, using a collection of quality criteria we systematically specify. We illustrate the viability of our approach by lifting out two particular aspects in which the current LOD Cloud can be immediately improved by automated means. Since not all quality aspects can be addressed algorithmically, we also give an overview of problem areas that may steer future endeavours in tooling, training, and best practices.

1. Introduction

In this article we investigate the quality of literals in the Linked Open Data (LOD) Cloud. A lot of work has focused on assessing and improving the quality of Linked Data. However, the particular topic of literal quality has not yet been thoroughly addressed. The quality of literals is particularly important because (1) they provide a concise notation for large (and possibly infinite) values spaces and (2) they allow text-based information to be integrated into RDF's graph based data model.

Our approach consists of the following steps. First, we create a toolchain that allows billions of literals to be analyzed. The toolchain is made available as Open Source code to the community and is integrated in two state-of-the-art data quality services: Luzzu and LOD Laundromat. Secondly, we use this toolchain to *analyze* the overall quality of literals on the LOD Cloud. Thirdly, based on the previous step, we give concrete suggestions for *improving* the quality of literals. Fourthly, we implement and evaluate some of these suggestions and show that our approach and toolchain are well-suited for implementing quality analysis and improvement on a Web scale. Our approach is different from existing work on quality assessment and improvement because we analyze and improve quality aspects on a very large scale and we integrate our results into existing tooling for quality assessment and data cleaning.

This paper is structured as follows. Section 2 discusses related efforts on quality assessment and improvement. In Section 3 we give our motivation for performing this work. In Section 4 we define a set of quality criteria for literals. The following Section describes the toolchain and its role in supporting the defined quality criteria. Section 6 reports our analysis in terms of the quality criteria defined in the previous section. In Section 7 we enumerate opportunities for improving the quality of literals based on our observations in the previous section. We implement two of those opportunities and evaluate their precision and recall. Section 8 concludes the paper and discusses further opportunities for research on literals quality.

2. Related work

Quality assessment for Linked Data is a difficult and multifarious topic. We purposefully focus on only a relatively isolated and restricted part of quality: the syntactic, semantic and linguishtic aspects of literal terms. Many publications have quantified various aspects of Linked Data in so-called 'observatory' studies. However, these studies have not included anything but a cursory analysis of RDF literals. A formal characterization of problem categories for data quality has been developed by [18]. These categories are not specifically targetted or tuned to fit Linked Data, but we use somewhat similar distinctions in our this chracterization in Section 4, Many vocabularies have been created that can be used to express data quality in, e.g., [12]. Currently a W3C Working Group is developing a standard vocabulary for expressing Linked Data quality.[1].

A test-driven approach towards Linked Data quality assessment has been developed by [15]. By using SPARQL query templates this allows the set of quality metrics to be easily extended. Another SPARQL-based approach is developed in [11]. Some aspects of quality are highly subjective and cannot be determined by automated means. For this [1] have presented a crowdsourcing approach towards quality assessment. Quality is not a static property of data, but something that can change over time as data gets updated. The dynamic aspects of data quality are observed in [16].

[1]See `http://www.w3.org/TR/vocab-dqv/`

3. Motivation

Literals are an important syntactic and semantic component of the Semantic Web's data model. They provide a concise notation for infinite value spaces such as (natural) numbers. The cute Linked Open Numbers dataset [22] shows that it simply does not make sense to assign an IRI to *everything*. Some things are better expressed in an abstract and possibly infinite value space with well-defined orderings. For instance, the ordering relation between the natural number is better expressed intentionally, by a short and simple function definition, than extensionally, by an inifinite number of pairs.

The second main benefit of literals is that they allow linguistic or text-based information to be expressed in addition to RDF's graph-based data model. While IRIs are (also) intended to be human-readable [9], there is an advantage to the use of literals containing natural language strings in order to convey human-readable infromation about resources. Also, in some datasets IRIs are intentionally left opaque as the human-readability of universal identifiers may negatively affect their permanence [4]. Since the Semantic Web is a universally shared knowledge base, natural language specifiers are particularly important in order to ease the human processability of information in different languages.

Assessing the quality of literals for each dataset is an important ingredient for assessing the overall quality of a dataset. Specifically, indicators of literal quality can be fed into Luzzu [8], a state-of-the-art quality assessment framework. Secondly, a LOD Cloud-wide overview indicates what are the current problems for the consumption of literals and can point to areas where data publication practices can be improved. Specifically, quantified quality indicators can be used in order to improve the cleaning process of the LOD Laundromat [3], a price winning data cleaning and republishing Web Service. We believe that it is best to base tomorrow's tooling, training, best practices and future standards on an empirical overview of today's problems.

Improving the quality of literals has (at least) the following concrete benefits for the consumption of Linked Data:

Efficient computation
If a data consumer wants to check whether literals l_1 and l_2 are identical she first has to perform the appropriate mappings. This is especially cumbersome in certain use cases in which the data consumer wants to check whether a given literal appears in a (possibly large) collection of RDF statements. However, if the data consumer can rely on the fact that all lexical expression are canonical with respect to their associated datatype, then she is able to perform literal indentity checking based on a simple, character-for-character string similarity check.

Data enrichment The availability of reliable language tags that indicate the language of a textual string is an enabler for data enrichtment. Similarity metrics of literals are an important part of existing instance matching approaches [13]. If the language tags of strings are known then this allows notions of similarity to be defined that move beyond (plain) string similarity and that utilize richer linguistic means like natural language translation and lexical relationships between words such as "is synonymous with". Notice that this will even benefit monolingual datasets by pulling in statements about the same concept based on translations/synonyms, where the language-tagged string in a foreign language is merely a means for enriching the data.

User eXperience Knowing the language of user-oriented literals such as `rdfs:label` or `dc:description` helps to improve the User eXperience (UX) of Linked Data User Interfaces. Provided the language preference of the user is known or can be dynamically assessed, an application can prioritize language-compliant literals in the display of user-facing literals. Similar remarks apply to the approach of "value labeling" [21], in which natural language labels are used to denote resources rather than their resource-denoting IRIs. Finally, the canonicalization of literals can result in more readable lexical expressions overall (e.g., decimal "`01.0`" is canonicalized to "`1.0`"). While the data publisher may have intended to display literals in a certain serialization format, the utility of intended formats is application-specific and should therefore not be considered a good approach in Linked Data where unanticipated reuse is a major goal.

Natural Language Processing Various NLP tasks are shown to benefit from background knowledge available in the LOD Cloud. Examples of such tasks are Named Entity Linking, Entity Coreference, Event Coreference and Sentiment Analysis. To facilitate the use of semantic knowledge these tasks need to link natural language text to Semantic Web resources. This is generally a two-stage process: Firstly, a natural lagnuage expression is linked to relevant Semantic Web literals, e.g., by using a full-text index like LOTUS [14]. Secondly, the literals from the previous step are linked to IRIs. The second step is straightforward and directly follows from the data definition by the data creator. The first step of matching textual expressions to relevant literals is more challenging, since a relevance metric has to be calculated based on structured data and meta-information. The improvement of literal quality, specifically the availability of reliable language tags, results in more reliable relevance metrics. E.g., a relevance metric may favor literals whose language tag matches the text-based query. In addition, reliable datatype information can be used to distinguish $\langle$`"007"`, `dt:kilo`$\rangle$ (the weight of a bag of potatoes) from $\langle$`"007"`, `xsd:string`$\rangle$ (the name of a fictional character).

4. Specifying quality criteria for literals

4.1. Syntax of literals

We define the set of literal terms as $L := (IRI \times LEX) \cup (\{\texttt{rdf:langString}\} \times LEX \times LTAG)$, where IRI is the set of Internationalized Resource Identifiers as per RFC 3987 [10], LEX is the set of Unicode strings in Normal Form C [6], and $LTAG$ is the set of language tags as per RFC 5646 [20]. Literals that are triples are called **language-tagged strings** LTS. The first element of a literal is called its **datatype IRI**, the second element is called its **lexical expression**, and the third element – if present – is called its **language-tag**. Syntactic RDF formats sometimes allow literals to be denoted by only a lexical expression lex. Such literals have previously been called **simple literals** and are mere abbreviations of the [air $\langle \texttt{xsd:string}, lex \rangle$.

When we talk about a given collection of data we often want to lift out those specific literal terms that appear within it. Let O_G denote the set of object terms that appear in a given graph G, defined as $O_G := \{o \mid \langle s, p, o \rangle \in G\}$. The literal terms that appear in G are then defined as $L_G := L \cap O_G$. Similarly, the literal terms that appear in a given dataset $H = \langle G_0, \langle i_1, G_1, , \rangle \ldots, \langle i_n, G_n \rangle \rangle$, with default graph G_0 and named graphs G_1 through G_n, are defined as $L_H := \bigcup_{i=0}^{n} L_{G_i}$.

4.2. Semantics of literals

The meaning of RDF data is determined by an interpretation function I that maps RDF terms to resources, denotes by IR, and RDF statements to the binary truth values, denoted by 0 and 1. Not every IRI denotes a datatype. Let us call the subset of resources that are datatypes D. The datatype IRIs are then defined by $IRI_D := \{i \mid i \in IRI \wedge I(i) \in D\}$. Not every lexical expression is allowed to occur in combination with every datatype. More specifically, every datatype $d \in D$ specifies a subset of the lexical expressions, denoted $LEX(d)$, that are syntactically valid to appear in a literal pair or tuple together with a datatype IRI i for which $I(i) = d$.

On the Semantic Web the meaning of an RDF term is the resource it denotes. We first consider the meaning of datatype IRIs. Every datatype IRI $i \in IRI_D \setminus \{\texttt{rdf:langString}\}$ denotes a datatype $d \in IR$. Every datatype d defines the following four things:

1. A set of syntactically well-formed lexical expressions $LEX(d)$ called the **lexical space** of d.
2. A set of resources $VAL(d)$ that can be denoted by literals of that datatype, called the **value space** of d.
3. A functional **lexical-to-value mapping** $l2v(d) : LEX(d) \to VAL(d)$. The resource that is denoted by a literal l is called its **value** or, symbolically, $I(l)$.
4. A (not necessarily functional) **canonical value-to-lexical mapping** $v2l(d)$.

The denotation of literal terms is determined by the partial mapping $IL : LIT \to IR$ (definition 1). IL is partial because a lexical expression may not belong to the datatype's lexical space. Which resource is denoted by which literal is determined by the combination of a specific datatype IRI i and lexical expression lex. Notice that the use of $I(i)$ is required in definition 1 in order to consider cases in which i denotes the same datatype as `rdf:langString` while not being the exact same term.

Definition 1 (Literal value)**.**

$$IL(l) := \begin{cases} lex & \textit{if } l = \langle lex \rangle \\ \langle lex, lc(tag) \rangle & \textit{if } Cond_A(l) \\ l2v(I(e))(lex) & \textit{if } Cond_B(l) \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

where

$$Cond_A(l) \Leftrightarrow l = \langle \texttt{rdf:langString}, lex, tag \rangle$$
$$Cond_B(l) \Leftrightarrow l = \langle e, lex \rangle \wedge lex \in LEX(I(e)) \wedge I(i) \neq I(\texttt{rdf:langString})$$

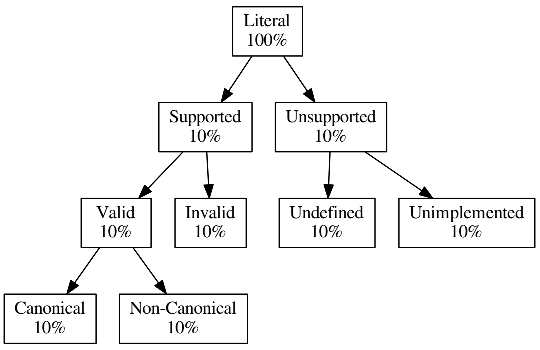
RDF processers are not required to recognize datatype IRIs other than `rdf:langString` and `xsd:string`. Literals with unrecognized datatype IRIs are semantically treated as unknown names. An RDF processor that recognized more datatypes is therefore not "more correct" but it is able to distinguish and utilize more subtleties of meaning.

4.3. Measures for literal quality

We want to distinguish between different aspects of literal quality. We start with the quality categories that relate to the interpretability and processability of literals. These categories, shown in Figure 1, are related to the four components a datatype specification should ideally contain as enumerated in Section 4.2.

A literal lit is *undefined* if its IRI i does not denote a datatype in D. Whether an IRI denotes a datatype or not is not specified in the RDF standards. We say that an IRI is defined iff lookup of the IRI leads to either (A) a machine-processable specification, (B) a human-readable formal specification, or (C) a human-readable informal description that can be unambiguously turned into a formal specification. For instance, the XSD datatype IRIs point to the XML Schema 1.1 Part 2: Datatypes specification, which includes (A) and (B). We require the specification to include a lexical space, a value space and a mapping between the two.

Figure 1. Quality categories for datatype support in RDF literals.



If a specification is only missing a canonical value-to-lexical mapping we include its literals under the 'non-canonical' category (see below). The only exception to these criteria is `rdf:langString` which does not have a lexical space (and therefore no mappings either).

A literal is *unimplemented* if its IRI denotes a defined datatype but is not implemented in a specific RDF processor. A literal is *unsupported* if it is either undefined or unimplemented. A supported literal is *valid* if its lexical expression can be mapped by $l2v(d)$ to a value. Valid literals are of higher quality than invalid or unsupported ones because they expose more meaning, i.e., they are not treated as unknown names.

A literal lit is *canonical* iff there exists a canonical value-to-lexical mapping $v2l(d)$ for its datatype d and $lit = v2l(l2v(d)(lit))$. Canonical literals are of higher quality than non-canonical ones because they allow identity to be assessed more efficiently.

As is apparent from the foregoing, language-tagged strings are to be handled in a special way. They define a value space but no lexical space (since language tags are expressed external to the lexical expression). Language tags must follow the grammar given in RFC 5646 [20] and the constituting subtags must match registrations inside the IANA Language Subtag Registry[2].

[2]Downloaded from `http://www.iana.org/assignments/language-subtag-registry/language-subtag-registry`

For language-tagged strings we can define multiple quality criteria. Firstly, the language tag may be malformed. Secondly, a well-formed language tag may not be registered in the IANA language tag registry. Thirdly, some registered language tags are out of date. Fourhtly, the language tag may be well-formed, registered and up-to-date, but it may not denote the (primary) language of the lexical expression.

Another aspect of the linguistic quality of literals is that linguistic content is often encoded in other datatypes, mainly `xsd:string`. We expect that there is a wealth of linguistic content encoded in today's Semantic Web that cannot be effectively used to serve users. In addition, there are use cases for the integration of NLP tooling, combining structure and un- or semi-structured data, that would benefit from knowing the language tag of a given string.

5. Implementation

In this section we describe the data and software we use for our analysis and for the algorithmic improvements we make.

5.1. LOD Laundromat data collection

The analysis as well as the evaluation of the improvement modules is conducted over the LOD Laundromat [3] data collection, which currently consists of about 650K data documents and 38 billion ground statements. The data is collected from data catalogues (e.g., Datahub) and contains datasets that users have uploaded through the Web API[3]. The LOD Laundromat collection contains approximately 12.38 billion literals and already stores the following metadata properties about literals for each of its datasets:

- The number of distinct literals.
- The number of literal occurrences (possibly containing duplicate occurrences).
- The minimum, maximum, average, median and standard deviation of the literal lengths.

[3]See `http://lodlaundromat.org/basket`

Since the LOD Laundromat only includes syntactically processable statements it is missing all literals that are part of syntactically malformed statements. The reason for this is that whenever a statement is syntactically malformed it is impossible to reliably locate whether a literal terms is present and, if so, where it occurs. For example, the syntactically malformed line [0] inside a Turtle-family document may be fixed to a triple [1], a quadruple [2] or two triples [3].

```
[0]   <a> <b> "c, d>    .
[1]   <a> <b> <c,d>     .
[2]   <a> <b> "c,"  <d> .
[3]   <a> <b> "c" , <d> .
```

The absence of well-formed literals that appear within malformed statements does not influence the meaning of an RDF graph or dataset. A statement must (at least) be syntactically well-formed in order to be interpretable. RDF semantics describes meaning in terms of truth-conditions at the granularity of statements: $I(\langle s, p, o\rangle) = 1 \Leftrightarrow \langle I(s), I(o)\rangle \in \mathit{IExt}(I(p))$, where IExt is the extension function mapping resources (called properties) to pairs of resources. While a literal can have a reference or denotation of its own, that denotation does – by itself – not express a basic thought or proposition. Paraphrasing Frege, it is only in the context of a triple that a literal has meaning.

5.2. *Toolchain*

While all LOD Laundromat data can be accessed through open Web APIs, we have used the following dedicated tools that we have developed to support running large-scale experiments over LOD Laundromat data. All our tools are (of course) published as Open Source software or as Web Services to the community.

Frank **[2]** A command-line tool that allows data to be streamed at the level of singular triple pattern fragments.

LOD-Laundromat-API[4] A SWI-Prolog[5] library that allows LOD Laundromat data and services to be accesses from whithin the ClioPatria triple store and Semweb library [23].

plRdf[6] A SWI-Prolog library that implement generators and parsers for the primitive XSD datatypes as well as several other datatypes that occur in RDF data.

For the assessment and improvement of language-tagged strings we use three existing state-of-the-art Automatic Language Detection (ALD) libraries:

1. Apache Tika - We use a NodeJS wrapper[7] for the 1.10 version of Apache Tika[8]. Apache Tika constructs a language profile of the text to detect and compares it with the profile of the set of known languages. The profiles of these languages are collections of texts which should be representative for the usage of those languages in practice. Such language profile is called corpus. Corpus accuracy depends on the profiling algorithm chosen (word sets, character encoding, N-gram similarity, etc.). Apache Tika uses 3-gram similarity as such three-word groups are useful in most practical situations. According to the documentation, this algorithm is expected to work accurately with short texts. Tika can detect 18 languages (17 languages with European origin and Thai language).
2. CLD (Compact Language Detection) library - The NodeJS CLD library[9], which is built on top of Google's CLD2 library[10]. The original library recognizes text in 83 languages, while the NodeJS wrapper detects text in over 160 languages. CLD is programmed as a Naive Bayesian classifier which chooses one of the three possible algorithms: based on unigrams, on quadrams or defined by the script itself. This library makes use of hints supplied by the user, on encodings, expected language or domain URL.
3. Language-detection[11] (abbreviated as LangDetect or LD) is a library developed by Nakatani Shuyo in Java. A commonly used plugin for language detection in ElasticSearch[12] is based on this library. This library uses 3-gram similarity metric and a Naive Bayesian filter. The language profiles (corpora) used by the library have been generated from Wikipedia abstracts.

[5]See `http://www.swi-prolog.org`

[7]`https://github.com/ICIJ/node-tika`
[8]`http://tika.apache.org/1.10/index.html`
[9]`https://github.com/dachev/node-cld`
[10]`https://github.com/CLD2Owners/cld2`
[11]`https://github.com/shuyo/language-detection`
[12]`https://github.com/jprante/elasticsearch-langdetect`

The reported precision of the language-detection library is 99.8% for 53 languages.

The chosen ALD libraries are reportedly widely used (for e.g. in ElasticSearch) and characterized with remarkable accuracy on the supported languages and text sizes. Although the chosen set still remains – to some extent – arbitrary, note that it is trivial to include more libraries as one sees fit.

5.3. Luzzu Framework

Luzzu[13] [8] is a quality assessment framework for Linked Data. The rationale of Luzzu is to provide an integrated platform that: (1) assesses Linked Data quality using a library of generic and user-provided domain specific quality metrics in a scalable manner; (2) provides queryable quality metadata on the assessed datasets; (3) assembles detailed quality reports on assessed datasets. Furthermore, we aim to create an infrastructure that:

- can easily be extended by users by defining custom, domain-specific metrics;
- implements quality-driven dataset ranking algorithms facilitating use-case driven discovery and retrieval.

Two quality metrics were implemented and used within Luzzu[14]: (i) to assess the validity of a datatype vis-a-vie their lexical value; and (ii) to assess the correctness of a string's language tag.

5.3.1. Assessing the Datatype's Compatibility

In this metric we assess the compatibility of a datatype literal against its lexical value. For example whilst `"10"^^xsd:int` is correct, a value `"10"` with a datatype defined as `xsd:dateTime` is incorrect. Given $T_{literals}$ as the total number of literals, and $T_{correctLiterals}$ total number of correctly defined literals, a quality value (Q_{dt}) is calculated as follows:

Definition 2. $Q_{dt} = \frac{T_{correctLiterals}}{T_{literals}}$

A literal is a *correct literal*, if its lexical value is not malformed or ill-typed, or if the literal is not undefined.

5.3.2. Assessing the Correctness of a Language Tag

This metric determines whether the language tag used in an RDF Literal is the correct one, in terms of (i) its tag syntax (following the RDF 1.1 Concepts[15]) and (ii) also in terms of actual language of the word (or description). For example, `"bread"@en` and `"ħobż"@mt` have the correct language tag, on the other hand `"ħobż"@en` or `"ħobż"@maltese` should be flagged as incorrect.

For simple single word literals we use the Lexvo [7] service [16]. Lexvo is an enriched linguistics knowledge base that interconnects entities to each other (for example different meanings and translations of a word) and also to entities on the Web of Data. Given a string literal with its corresponding tag, a request to the Lexvo API is made and a dereferenceable RDF resource is returned. This resource is then queried and if a `rdfs:seeAlso` is found, then we deem a string literal to have the correct tag.

In order to identify the correctness of a language tag in a multi-word literal (e.g in a description) we use the Xerox Language Identifier[17]. We experimented with the service for its identification correctness by providing various sentences in English, Italian and German. The service always returned the correct identification. On the other hand, this cannot be considered as a guarantee for all languages. The authors in [17] state that the service's identification correctness is not guaranteed for certain languages.

The result of this metric is dependent of these two services, therefore we consider this metric to give us an estimate. The quality value (Q_{lt}) is calculated as follows:

Definition 3. $Q_{lt} = \frac{T_{correctLanguageTags}}{T_{stringLiterals}}$

[13]Sources: `https://github.com/EIS-Bonn/Luzzu`; Website: `http://eis-bonn.github.io/Luzzu/`

[14]All metric implementations are external to Luzzu, therefore the results obtained are through the metrics imported to the framework.

[15]`http://www.w3.org/TR/rdf11-concepts/`

[16]`http://lexvo.org`

[17]`https://services.open.xerox.com/bus/op/LanguageIdentifier/GetLanguageForString`, an external service that tries to identify the language of a given sentence

where $T_{correctLanguageTags}$ is the number of correct language tags attached to a string literal and $T_{stringLiterals}$ is the total number of string literals.

6. Analysis

We have analyzed 1,457,568,017 literals from the LOD Laundromat data collection for the quality criteria described in Section 4.3. The full and up-to-date results are available online at `http://wouterbeek.github.io/quality`. We see that the vast majority of literals are valid and a modest majority of them are also canonical. However, 79% (or 1,108,813,673 occurrences) of literals are (plain) strings which have very few syntactic strictures. When we look at the more complex datatypes, e.g., dates, times and floating point numbers, we see that there is still a lot of room for improvement (see below).

Undefined Most IRIs that occur in the context of a literal come with neither of the three criteria we consider sufficient for an IRI to denote a datatype, as specified to Section 4.3. For instance, none of the DBpedia datatype IRIs is currently defined. Many datatypes that have some form of human-readable informal description do not provde enough information and/or clarity that would allow them to be implemented. An example of this is `sysont:Markdown`[18] which is described in Listing 1. This informal specification still lacks several key components: Firstly, the value space can either be defined as the set of Markdown-formatted strings or in terms of a formal abstraction of Markdown documents, similar to `rdf:XMLLiteral`'s DOM model. The grammar pointed to in the `rdfs:seeAlso` property is itself not formally specified. Finally, there does not yet exist a (generally accepted) canonical form for writing Markdown.

Listing 1: Informal description of Markdown datatype.

```
sysont:Markdown a rdfs:Datatype ;
  rdfs:comment "A string literal formated using
               markdown syntax." ;
  rdfs:label   "Markdown formated string" ;
  rdfs:seeAlso "http://daringfireball.net/
               projects/markdown/syntax" .
```

[18]`sysont` expands to `http://ns.ontowiki.net/SysOnt/`

Table 1
The five most occurring undefined datatypes.

Datatype IRI	Occurrences
`dt:second`	2,326,298
`dt:minute`	682,790
`dt:squareKilometre`	643,493
`dt:centimetre`	382,281
`dt:kilogram`	356,321

We notice that there is currently not a strong practice of defining datatypes in terms of XML Schema. In fact, we did not find such a definition outside of the original XSD specification. Also, while there is no inherent reason why an informally specified datatype should be ambiguous and/or incomplete, in practice we have not found an informal description that is unambiguous and complete. Table 1 shows the most often occurring undefined datatypes. The vast majority of these datatypes are defined in DBpedia.

1457568017

Partially defined Some datatypes are partially defined. For instance `http://purl.org/dc/terms/W3CDTF` does not define a canonical mapping, but does allow multiple lexical expressions to denote the same value. For instance the time zone designator is defined as "`Z` or `+hh:mm` or `-hh:mm`", with `hh` and `mm` ranging from 0 to 23 and 59 respectively. This allows `+01:00` and `-23:00` or `+00:00` and `-00:00` to express the same value. We also note that a canonical mapping is sometimes hard to specify and may sometimes not be very useful. An example of this is `rdf:HTML` which does not specify a canonical mapping, which would have to map DOM models to HTML serializations.

Invalid Table 2 shows the datatypes that have the highest number of invalid literals. Overall, only 0.11% of the literals are invalid. However, as was mentioned before, 79% of all literals are (language-tagged or plain) strings for which almost every lexical expression is valid. As soon as the data gets more compliated, the percentage of invalid occurrences goes up.

Non-canonical Table 3 shows the eight datatypes with the highest number of non-canonical literals. Overall, 3.5% of the literals are non-canonical. Again, the strings are canonical by definition, since they map onto themselves. On the other hand, the majority of the floating-point numbers (either `xsd:dobule` or

Table 2
The five datatype IRIs with the highest number of invalid literals.

Datatype IRI	**Occurrences**
`xsd:int`	511,741
`xsd:decimal`	122,738
`xsd:dateTime`	98,505
`xsd:gYearMonth`	16,469
`xsd:gYear`	11,957

Table 3
The eight datatype IRIs with the highest number of non-canonical literals.

Datatype IRI	**Occurrences**
`xsd:float`	30,152,304
`xsd:double`	17,783,414
`xsd:decimal`	2,127,133
`rdf:XMLLiteral`	245,457
`xsd:dateTime`	224,994

`xsd:float`) are non-canonical. The reason for this is that their canonical format is quite specific: it must always be written in scientific notation with the exponent sign `E`. For instance ,this means that a floating point number “1.0” must be canonically written as “1.0E0”.

6.1. Language-tagged strings

The LOD Laundromat currently contains 5.31 billion natural text expressions, out of which 2.74 billion (51.66%) have a language tag specified by the data creator. This means that around half (48.34%) of the LOD Laundromat natural text literals do not have associated language tag. Furthermore, we looked into the distribution of the encountered language tags (Table 4). By far the most literals (over 1 billion) are tagged as English literals, followed by several other languages of European origin: German, French, Italian and Spanish. 70.33% of the defined language tags describe literals in one of the 10 most frequent languages.

6.2. Quality Analysis using the Luzzu Framework

In order to illustrate that the here presented toolchain integrates well with existing quality frameworks, we have also run initial experiments on the Luzzu framework for quality analysis [8]. We used Luzzu in order to process 470 data document from the LOD Laundromat collection. For each of these documents Luzzu calculated the *compatible datatype metric* and the *correct language tag usage metric*. For these specific documents Luzzu determined that, on average, 70.44% of the RDF string literals in LOD Laundromat have a correct language tag. On the other hand, only 38.69% of RDF literals have a compatible datatype.

Table 4
The distribution of language tags as calculated over 12,380,443,617 literals.

Language tag	**Occurrences**
en	1,049,037,147
de	165,996,755
fr	149,507,401
it	126,550,182
es	89,464,945
ru	84,663,662
nl	81,413,963
pl	67,271,847
pt	61,105,515
ja	53,954,942
other	813,744,170
no tag	2,577,080,307
text literals	5,319,790,836
all literals	12,380,443,617

We inspected a sample of documents whose quality value was less than 40% for both metrics. Starting from the *correct language tag usage* metric, we group problematic (as opposed to correct) triples in the following categories:

– Literal values without a language tag;
– Literal values with non-alphabet symbols (e.g `"related_software"@en`);
– Literal values with syntactic errors (e.g. `"flow cytometer sorter"@en`);
– Literal values with an unknown language tag (e.g `"article"@en-US`).

The majority of problematic triples fall in the first category. The third and fourth categories are the most interesting. The former category deals with literals that were identified incorrect by the metric’s external service due to some language syntactic flaw. For example, in `"flow cytometer sorter"@en` the term *cytometer* should have been written as *cytometry*. The final category deals with the actual syntax expected of the language tag. Although the tag `@en-US` is correctly defined as per the BCP47 standard [20], our metric expects two/three letter language tag as defined by the Linked Languages Resources[19].

On the other hand, the *compatible datatype metric* was more simple to analyse as most of the literals lacked a datatype. Figures 2 and 3 show a subset of the documents that were analysed and their quality. All quality results can be visualised at `http://jerdeb.github.io/LODLaundromatCrawler`.

7. Improvement

In this section we discuss opportunities for improving the quality of literals on the LOD Cloud. We base this discussion on the results of the analysis in the previous section. Some quality aspects can only be improved by the original data creator and/or publisher. While we cannot automate these improvements, we can give suggestions and point to the major pain points based on empirical observation rather than intuition. In order to show that our toolchain can indeed be used to scale quality improvement, we lift out two quality improvements that can be automated and that support two of the use cases in Section 3: (1) Efficient computation for equivalence tests, by automatically converting non-canonical valid literals to canonical ones. (2) Natural language processing by automatically assigning language tags to textual literals that did not have language tags before.

7.1. Improving datatypes

Undefined $\rightarrow$ defined Undefined literals can only be improved by adding definitions that take all aspects of Section 4.2 into account. This may be partially automated based on the lexical expressions that appear in the context of a given IRI. However, merely looking at the associated lexical expressions is not enough. For instance, the fact that the values "1", "203", "9009" appear in the data does not tell us whether the datatype is an `xsd:positiveInteger` or an `xsd:nonNegativeInteger`. Deciding on the most general primitive datatype, `xsd:decimal` in this case, also does not suffice since, for this particular example, `xsd:gYear` would apply just as well. The problem becomes even more complex when non-standard datatypes are considered, which could map lexical expression "1" to the Mount Everest and lexical expression "203" to afternoon sunsets. The analysis in Section 6 gives an overview of the size of this quality issue and hints at actions that may be undertaken to improve quality in this respect (e.g., defining DBpedia datatypes in terms of the XSD primitives).

Unimplemented $\rightarrow$ implemented Unimplemented literals can only be improved upon by adding support to a particular RDF processor. At the moment, only few datatypes beyond the XSD primitive types are implemented by any processor. Section 6 shows that it is not easy to implement most datatypes since many of them are underspecified. Improving this may require a more structured way of defining a datatype, something the current standards do not cover. Also, while implementing *plRdf*, we discovered that it helps to cross-validate against another library (in our case Sesame 4). In a similar way, we hope that the availability of *plRdf* will make it easier for other to add support for more datatypes in their RDF processors.

7.2. Improving lexical expressions

Invalid $\rightarrow$ valid Invalid literals can only be improved upon by the original data publisher. We cannot automate this task as it requires us to choose between changing the datatype IRI to match the lexical expression, changing the lexical expression, or changing both. We can however give a list of the often occurring mistakes that data creators should be aware of. Based on our empirical observations these are the top 5 mistakes, along with suggestions of how to avoid them:

1. `xsd:int` is not the same as `xsd:integer`. The former is a short integer and cannot be used to express integers smaller than -2147483648 or larger than 2147483647.
2. RDF IRIs are case sensitive [5]. Specifically `xsd:datetime` is not the same as `xsd:dateTime`.
3. `xsd:date` must not include a temporal specifiers. `xsd:dateTime` is used for this instead.
4. Datatype IRI are regularly not resolved with respect to their RDF prefixes.

[19]`http://linkedvocabs.org/lingvoj/`

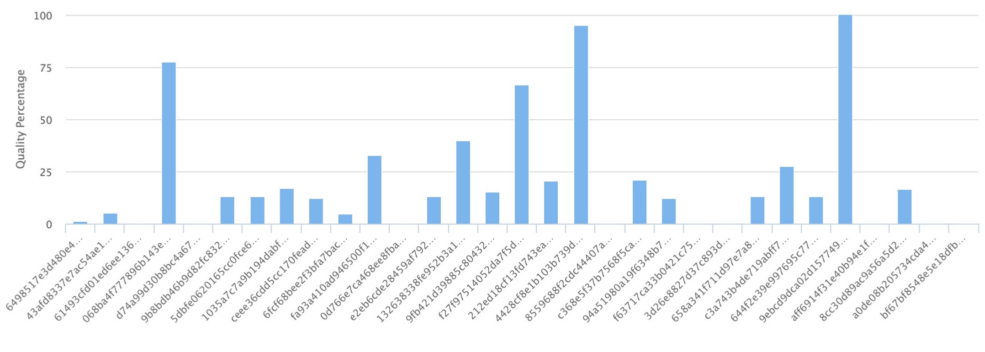


Figure 2. Quality Assessment over a **subset** (around 30 documents) of LOD Laundromat for Compatible Datatypes

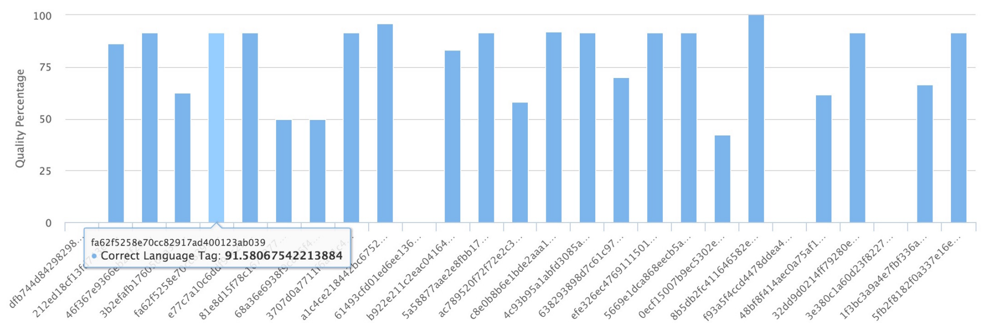


Figure 3. Quality Assessment over a **subset** (around 30 documents) of LOD Laundromat for Correct Language Tag Identification

Non-canonical → canonical Canonical literals provide a significant computational benefit over non-canonical valid literals for several use cases. For instance, checking whether two terms or statements are identical or not no longer requires parsing and generating, i.e., string similarity suffices where one whould have to calcuate $v2c(l2v(l_1)) \equiv v2c(l2v(l_2))$ otherwise.

The improvement of non-canonical literals is conditional on other quality improvements. Firstly, the datatype has to specify a canonical mapping. Secondly, there has to be a tool that implements this mapping. When these two preconditions are met we can algorithmically generate canonical out of non-canonical literals. We have done this for **{ToDo** Wouter**}** literals. We have compared the results of our *plRdf* library to *Sesame 4*, to the extent that there is a single canonical form. Specifically, the canonical form for XSD doubles and float allows some deviation according to the specification [19]. While we have implemented several other often occurring datatypes, e.g., `dct:W3CDTF` and `dct:RFC4646`, it is not easy to check our implementation's correctness since very few comparable implementations exist.

7.3. Improving language tags

Invalid → valid As is the case with lexical expressions, invalid language tags can only be improved by the original data publisher.

Outdated → up-to-date Outdated language can be automatically updated, since IANA registers the deprecation process of language tags.

No language tag → language tag We test tools to assign a language tag to textual literals without one. For this purpose we test language detection improvements on textual lexical expressions from the first 20,000 documents of the LOD Laundromat data collection. We define a textual lexical expression as a lexical expression of datatype xsd:string or xsd:langString which has at least two consecutive Unicode letters.

In an attempt to improve the language tags coverage of LOD Laundromat, we apply three language detection libraries (described in Section 5.2). We are interested in the following aspects. How often does the automatically detected language tag coincide with the user-assigned tag? How accurate are the language detection libraries? Does the accuracy of detection differ per primary language or for various string sizes? Are certain languages or string sizes easier for language detection? How often do the libraries refrain from assigning a language tag? Can we combine the libraries and thus improve the accuracy of language detection?

In our experiments, we primarily focus on the set of tagged literals. These contain a "golden" language tag, which allows us to easily evaluate the accuracy of our solutions. For these, we report the precision, recall and F1-value of each of the language detection libraries. We assume that the accuracy of the language detection on the language-tagged strings is comparable to the accuracy on textual lexical expressions with no user-defined language tag.

The language tag assigned by the users and by the automatic detection libraries can be of arbitrary length or complexity. Since our language detection tools provide an ISO 639-2 two-character language code in most of the cases, we focus our comparison on the initial two characters of each language-tagged string. This granularity of comparison is satisfactory for most cases, although in exceptional situations the secondary language tag can also describe the language. This is the case for Chinese languages where `zh-cn` denotes a different language than `zn-tw`.

The accuracy for each of the libraries over all language-tagged natural text is shown in Table 5. The highest precision and F1-score is achieved by the CLD library, which covers highest number of languages (160). LangDetect has best performance in terms of recall, while Apache Tika has lowest. It is interesting to notice that Apache Tika never returns an empty language tag, while CLD often gives no language suggestion.

We further investigate whether the accuracy of the libraries is sensitive to specific language tags or string sizes. The outcome of this analysis is shown in Table 6. Each of the cells in the Table represents an intersection of a language and string size bucket, while the values in the cell show the F1-accuracy of each of the three libraries: Tika, CLD and LangDetect, correspondingly. While CLD has highest accuracy in the majority of the cells, there are notable exceptions. For instance, when it comes to Italian text expressions or short French expressions, the accuracy of the LangDetect library is higher. Guided by these insights, we combine the libraries by applying the most accurate libraries per cell. As a result, the detection judgement by CLD is used for most cells (55.04%), while Apache Tika (28.82%) and LangDetect (16.13%) come second and third. By combining the libraries in such manner we are able to improve slightly the F1-value in comparison to the most successful library (see last row of Table 5).

Figure 4 shows the aggregated accuracy per bucket for each of the libraries. Note that there is no intersection of the plotted lines: for any bucket of text size, CLD has the highest F1-value, while Tika has the lowest. However, the text size does correlate with the general success of language detection (by any library). Concretely, short strings which contain only one word (bucket 0) or two words (bucket 1) are much harder to detect correctly than longer strings. On the other hand, expressions from bucket 8 (between 129 and 256 words) can be detected with almost perfect accuracy.

This tendency is confirmed for the most frequent 10 languages (Figure 5). Every data point represents an average F1-value over the three libraries for a given language and bucket. Libraries can successfully detect the language of sufficiently long literals (bucket 3 literals already have an F-measure of around 80%, growing to above 90% for bucket 4). Languages of Indo-European origin closely follow this distribution, while Chinese and Vietnamese have a different behavior: the accuracy depends much less on the length of text.

8. Conclusions

We predented our toolchain for large-scale data quality analysis and improvement, which extends the LOD Laundromat data cleaning and republishing architecture. We have focussed on the quality of literals, an area of Linked Data quality that has not been thoroughly investigated before. We have systematically specified a collection of quality criteria that are specific for RDF literals. We have shown that our toolchain is able to analyze data quality, in terms of those quality

Figure 4. Accuracy per language detection library

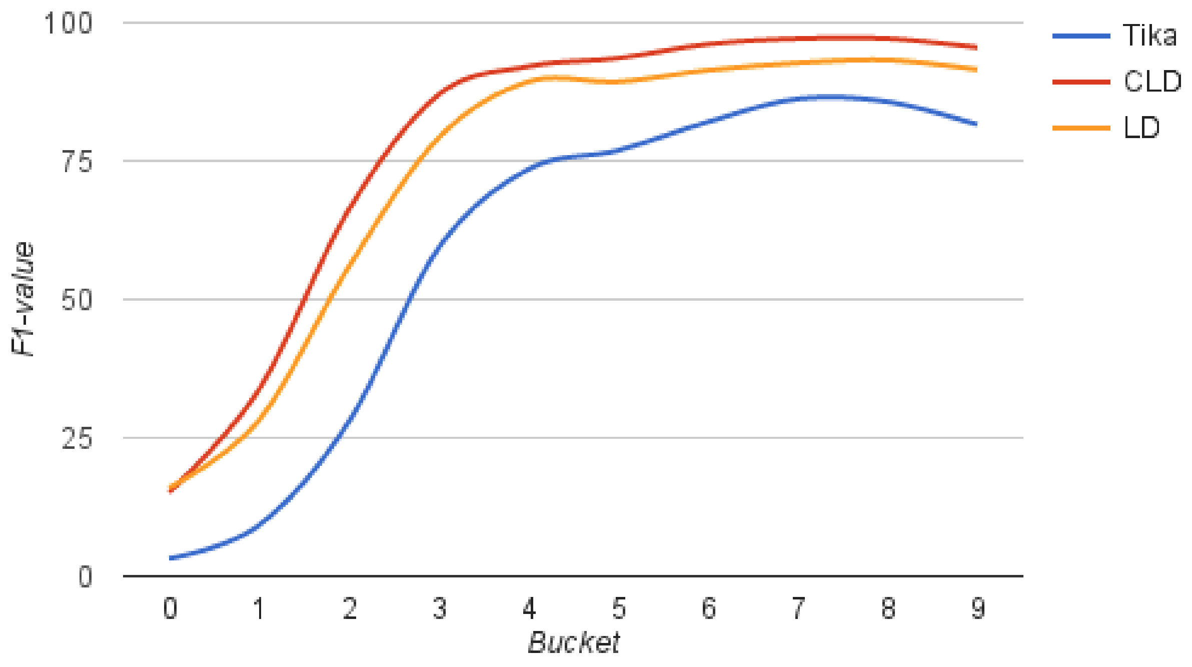


Figure 5. Accuracy per language tag for the 10 most frequent languages

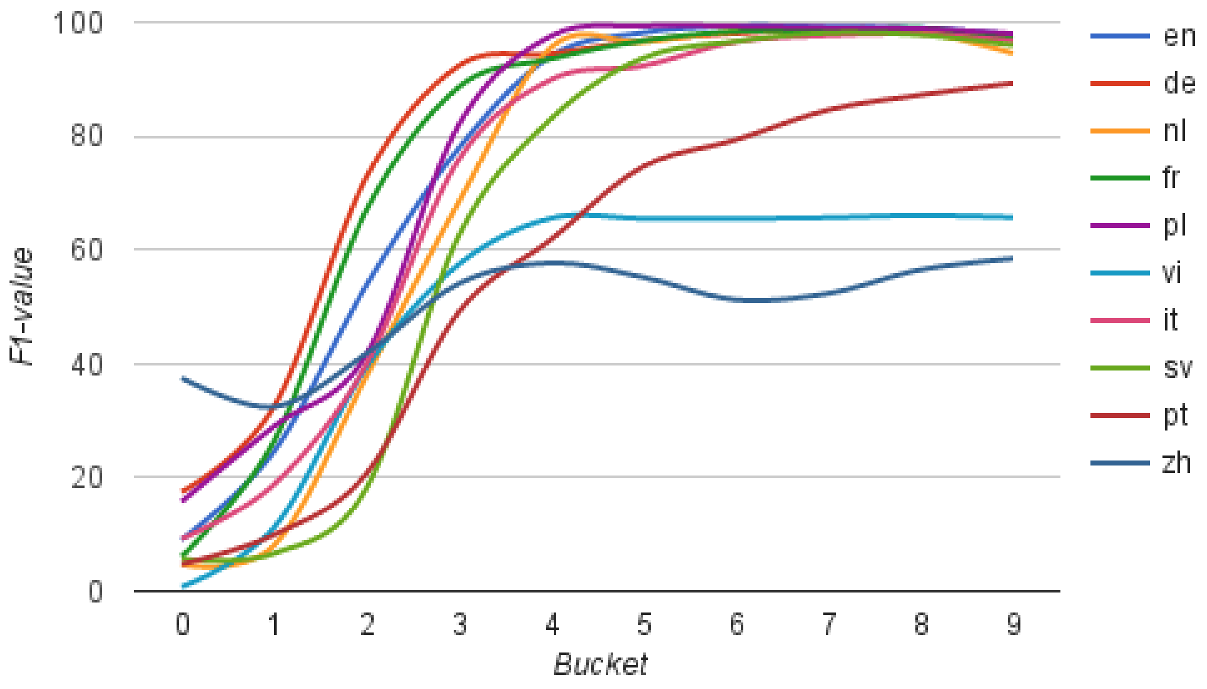


Table 5
Performance of the ALD solutions

Library	Precision	Recall	F1-value
Apache Tika	19.77%	19.77%	19.77%
CLD	75.55%	32.21%	45.16%
LangDetect	36.97%	35.45%	36.19%
Combined	59.53%	37.22%	45.80%

Table 6
F1-value accuracy of the libraries per bucket size and language

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
en	1.88	7.11	26.53	60.09	86.53	96.01	98.85	98.98	98.55	96.54	64.4	83.66	77.54	71.43	n/a
	12.76	39.55	76.85	93.07	99.24	99.64	99.78	99.56	99.33	98.62	90.31	94.11	93.33	91.43	n/a
	12.45	27.22	57.75	80.45	96.45	98.9	99.4	99.16	99.14	98.42	84.03	93.97	90.88	82.14	n/a
de	6.87	15.07	54.63	84.26	88.9	94.23	96.85	98.4	98.05	95.45	67.39	45.87	61.54	81.82	100.0
	14.1	33.52	80.06	96.96	96.74	97.75	98.54	99.13	98.9	98.02	84.9	86.67	92.31	100.0	100.0
	31.55	48.88	83.4	95.79	97.71	98.47	98.72	99.14	99.26	98.65	85.49	89.91	92.31	100.0	100.0
nl	1.39	4.83	21.01	66.15	96.53	97.15	98.8	98.65	97.84	92.39	62.92	49.72	41.18	16.67	0.0
	4.49	8.13	45.89	64.13	93.67	94.78	97.48	98.72	98.83	96.54	84.21	80.35	81.25	40.0	0.0
	8.25	11.19	46.22	74.77	96.66	97.63	98.92	98.81	98.26	94.74	75.77	67.96	47.06	16.67	0.0
fr	3.62	19.96	60.01	82.74	88.66	95.12	98.61	99.02	99.08	96.5	86.44	86.14	60.0	60.0	n/a
	4.93	24.32	67.08	90.0	96.17	97.01	98.54	99.06	99.19	97.31	91.0	90.1	60.0	75.0	n/a
	9.71	35.01	73.52	93.09	95.87	98.12	98.02	97.44	98.64	97.39	89.27	88.12	60.0	100.0	n/a
pl	11.98	23.24	35.61	75.02	96.86	99.29	99.28	99.05	98.82	97.97	93.69	76.39	41.67	100.0	n/a
	14.9	31.71	45.47	85.88	97.75	99.38	99.39	99.19	99.09	98.17	95.15	77.61	43.48	100.0	n/a
	20.51	32.12	43.54	85.02	98.15	99.36	99.21	98.93	98.76	97.88	93.17	77.78	41.67	100.0	n/a
vi	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	n/a	0.0
	0.02	14.33	61.44	87.06	98.51	98.2	98.45	98.65	99.2	98.88	90.15	89.3	46.15	n/a	100.0
	2.29	19.27	56.62	85.19	98.08	98.19	97.92	98.27	98.9	98.17	44.27	85.84	45.0	n/a	0.0
it	8.85	18.89	40.53	80.1	93.63	93.81	97.57	98.23	98.65	97.53	90.25	89.02	86.67	0.0	n/a
	1.45	9.44	29.94	62.07	81.93	89.85	94.44	96.44	96.77	95.6	82.81	53.37	53.21	0.0	n/a
	17.31	28.12	49.02	85.33	94.25	93.33	97.66	98.33	98.48	97.12	87.0	66.47	63.33	0.0	n/a
sv	0.97	2.31	12.56	49.28	73.23	89.12	94.35	97.12	96.83	94.5	78.89	52.17	0.0	0.0	n/a
	3.4	8.15	19.68	65.39	84.89	94.21	96.79	98.57	98.67	97.78	88.32	73.68	44.44	100.0	n/a
	12.61	9.41	21.9	72.64	91.12	97.8	98.76	98.71	97.82	95.66	84.43	60.87	40.0	100.0	n/a
pt	1.85	5.26	8.19	16.16	19.29	34.28	41.77	55.76	63.08	70.0	74.01	61.11	78.95	n/a	n/a
	7.47	12.72	27.5	64.33	82.0	94.88	98.04	99.17	99.45	99.18	98.71	88.55	100.0	n/a	n/a
	4.87	11.76	26.3	66.63	84.18	94.71	98.23	98.87	99.12	98.68	98.11	81.94	100.0	n/a	n/a
zh	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	n/a	0.0	n/a
	50.94	52.59	63.78	82.17	87.84	84.47	77.99	79.76	85.84	87.25	6.75	46.15	n/a	100.0	n/a
	61.6	44.77	61.5	79.95	85.2	81.02	75.69	77.2	83.59	88.06	83.91	64.29	n/a	12.5	n/a

criteria, on a very large scale. We have illustrated that our toolchain can be used by existing quality assessment frameworks, such as Luzzu. We have also pointed to areas where literal quality would be most effectively improved, because it is now possible to quantify the impact of data cleaning, and other quality improvement attempts, beforehand.

Finally, we have shown that it is possible to improve the quality of millions of literals (and thereby statements) very quickly, by algorithmic means. This does of course not apply to every quality criterion, e.g., it does not apply to subjective criteria. But at least the quanlity criteria that can be automatically improved *in theory* should now also be automatically improved *in practice*. We have shown that, when given state-of-the-art algorithms, our toolchain is able to improve the overall quality of the LOD Cloud in days, not decades.

References

[1] Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Sören Auer, and Jens Lehmann. Crowdsourcing linked data quality assessment. In *The Semantic Web–ISWC 2013*, pages 260–276. Springer, 2013.

[2] Wouter Beek and Laurens Rietveld. Frank: Algorithmic Access to the LOD Cloud. *Proceedings of the ESWC Developers Workshop 2015*, 2015.

[3] Wouter Beek, Laurens Rietveld, Hamid R Bazoobandi, Jan Wielemaker, and Stefan Schlobach. Lod laundromat: a uniform way of publishing other people's dirty data. In *ISWC 2014*, pages 213–228. 2014.

[4] Tim Berners-Lee. Cool uris don't change, 1998.

[5] Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 concepts and abstract syntax, 2014.

[6] Mark Davis and Ken Whistler. Unicode normalization forms, August 2012.

[7] Gerard de Melo. Lexvo. org: Language-related information for the linguistic linked data cloud. *Semantic Web*, page 7, 2013.

[8] Jeremy Debattista, Christoph Lange, and Sören Auer. LUZZU – a framework for linked data quality assessment, 2015.

[9] M. Duerst and M. Suignard. Internationalized Resource Identifiers, January 2005.

[10] Martin J. Dürst and M. Suignard. Internationalized resource identifiers (iris), January 2005.

[11] Christian Fürber and Martin Hepp. Using sparql and spin for data quality management on the semantic web. In *Business Information Systems*, pages 35–46. Springer, 2010.

[12] Christian Fürber and Martin Hepp. Towards a vocabulary for data quality management in semantic web architectures. In *Proceedings of the 1st International Workshop on Linked Web Data Management*, pages 1–8. ACM, 2011.

[13] Bernardo Cuenca Grau, Zlatan Dragisic, Kai Eckert, Jérôme Euzenat, Alfio Ferrara, Roger Granada, Valentina Ivanova, Ernesto Jiménez-Ruiz, Andreas Kempf, Patrick Lambrix, et al. Results of the ontology alignment evaluation initiative 2013. In *International Workshop on Ontology Matching, collocated with the 12th International Semantic Web Conference-ISWC 2013*, pages pp–61, 2014.

[14] Filip Ilievski, Wouter Beek, Marieke van Erp, Laurens Rietveld, and Stefan Schlobach. Lotus: Linked open text unleashed. In *COLD workshop, ISWC*, 2015.

[15] Dimitris Kontokostas, Patrick Westphal, Sören Auer, Sebastian Hellmann, Jens Lehmann, Roland Cornelissen, and Amrapali Zaveri. Test-driven evaluation of linked data quality. In *Proceedings of the 23rd International Conference on World Wide Web*, WWW '14, pages 747–758, New York, NY, USA, 2014. ACM.

[16] Tobias Käfer, Ahmed Abdelrahman, Jürgen Umbrich, Patrick O'Byrne, and Aidan Hogan. Observing linked data dynamics. In Philipp Cimiano, Oscar Corcho, Valentina Presutti, Laura Hollink, and Sebastian Rudolph, editors, *The Semantic Web: Semantics and Big Data*, volume 7882 of *Lecture Notes in Computer Science*, pages 213–227. Springer Berlin Heidelberg, 2013.

[17] N. Ljubesic, N. Mikelic, and D. Boras. Language indentification: How to distinguish similar languages? In *Information Technology Interfaces, 2007. ITI 2007. 29th International Conference on*, pages 541–546, June 2007.

[18] Paulo Oliveira, Fátima Rodrigues, and Pedro Rangel Henriques. A formal definition of data quality problems. In *IQ*, 2005.

[19] David Peterson, Shudi (Sandy) Gao, Ashok Malhotra, C.M. Sperberg-McQueen, and Henry S. Thompson. XML Schema Definition Language (XSD) 1.1 part 2: Datatypes, April 2012.

[20] A. Phillips and M. Davis. Tags for identifying languages, September 2009.

[21] Giovanni Tummarello, Richard Cyganiak, Michele Catasta, Szymon Danielczyk, Renaud Delbru, and Stefan Decker. Sig.ma: Live views on the web of data. *Web Semantics: Science, Services and Agents on the World Wide Web*, 8(4):355–364, 2010.

[22] Denny Vrandečić, Markus Krötzsch, Sebastian Rudolph, and Uta Lösch. Linked Open Numbers. In *Proceedings of RAFT 2010*, 2010.

[23] Jan Wielemaker, Wouter Beek, Michiel Hildebrand, and Jacco van Ossenbruggen. ClioPatria: A logical programming infrastructure for the Semantic Web. *Semantic Web Journal*, page to appear, 2015.