Gleaning Types for Literals in RDF Triples with Application to Entity Summarization

Kalpa Gunaratna[1(✉)], Krishnaprasad Thirunarayan[1], Amit Sheth[1], and Gong Cheng[2]

[1] Kno.e.sis, Wright State University, Dayton, OH, USA
{kalpa,tkprasad,amit}@knoesis.org
[2] National Key Laboratory for Novel Software Technology, Nanjing University, Nanjing, China
gcheng@nju.edu.cn

Abstract. Associating meaning with data in a machine-readable format is at the core of the Semantic Web vision, and typing is one such process. Typing (assigning a class selected from schema) information can be attached to URI resources in RDF/S knowledge graphs and datasets to improve quality, reliability, and analysis. There are two types of properties: object properties, and datatype properties. Type information can be made available for object properties as their object values are URIs. Typed object properties allow richer semantic analysis compared to datatype properties, whose object values are literals. In fact, many datatype properties can be analyzed to suggest types selected from a schema similar to object properties, enabling their wider use in applications. In this paper, we propose an approach to glean types for datatype properties by processing their object values. We show the usefulness of generated types by utilizing them to group facts on the basis of their semantics in computing diversified entity summaries by extending a state-of-the-art summarization algorithm.

Keywords: Type inference · Datatype properties · RDF triples · Feature grouping and ranking · Entity summarization · Dataset enrichment

1 Introduction

The rise of open data initiatives (e.g., Linking Open Data) has encouraged large-scale publication of data on the Web. Resource Description Framework (RDF) has been used extensively to encode information and publish as Semantic Web datasets and knowledge graphs. The direct consumers of these datasets, most of the time, are machines or software such as search and rank services. Therefore, it is imperative to enrich the datasets with additional information so that machines can interpret them properly. Assigning ontology classes as types to resources (typing) in RDF via the `rdf:type` property (which we refer to as semantic types) is one example of a data enrichment process. This can help machines to

identify similar or related resources by analyzing their types. Such enrichment can be exploited to improve the quality and reliability of datasets and facilitate analytics. Typing is performed on URI resources and, hence, only applies to object properties. Datatype property values are literals, and are usually associated with types by virtue of their syntactic data representation (which we refer to as syntactic types). For example, one assigns `datatypes` such as `xsd:string`, `xsd:integer`, and `xsd:date` to literals.

Syntactic types do not make explicit information that can be exploited for data analysis. However, the amount of information that datatype properties represent compared to object properties is significant in some real-world datasets. For instance, a recent version of DBpedia [10], which is one of the largest and most comprehensive encyclopedic datasets on the Web, has 1,600 datatype properties compared to 1,079 object properties[1]. Many of the literal values (other than noise) can be associated with types selected from a set of ontology classes that can promote proper semantic interpretation and use. For example, the property http://dbpedia.org/property/location has about 1,05,047 unique and simple literals that can be mapped to entities to infer the types (e.g., "California", "United States"). An entity is a thing (e.g., person, book, place) at the data level that encapsulates facts and is represented by a URI.

The importance of type information has been demonstrated by researchers in the Semantic Web community, including for inferring missing types for entities [13], ranking types for entities [17], and generating summaries using type graphs [18]. All these approaches make use of existing type information of "entities" or infer additional/missing types from them. There has not been any work related to inferring or computing types for literals in RDF/S datasets. In this work, we propose to address the issue of computing "semantic types" whenever possible for literal values of datatype properties. When types are available for datatype properties, they can be used in many interesting applications such as data integration, property alignment, and entity summarization. For example, in property alignment [5,7], we can utilize types to prune the candidates for alignment. Further, type prediction on datatype properties provides benefits similar to the works on type prediction for entities as in SDType [13]. We demonstrate the application of generated semantic types by extending the FACES entity summarization algorithm [6] and show how to group and rank features based on datatype properties. Our contributions in this work are twofold:

1. We analyze the object value of datatype properties and select a suitable class as the type from a given set of ontology classes.
2. We extend FACES (FACES-E) to group and rank both object and datatype properties to create entity summaries and demonstrate the usefulness of types to generate comprehensive entity summaries.

The rest of the paper is organized as follows. In Sect. 2, we analyze the problem of typing literals in datatype properties, and in Sect. 3, we define the problem and notions related to typing and summarization. In Sect. 4, we present

[1] http://wiki.dbpedia.org/About.

our typing algorithm and feature ranking for datatype properties, followed by evaluation in Sect. 5. We present a general discussion with future directions in Sect. 6, and present related work in Sect. 7. Finally, we conclude in Sect. 8.

2 Problem Analysis

Web Ontology Language (OWL) defines the types of properties: object properties that connect individuals to individuals and datatype properties that connect individuals to data values (literals)[2]. The object value of an object property is a URI that can be assigned an ontology class as its type via `rdf:type` property. But object values of datatype properties do not have ontology classes assigned as types and the only types available for them are the syntactic types referring to primitive, low-level implementation types. In this work, we try to suggest a class from a given set of classes as the type of the literal of the datatype property. See Fig. 1 which shows two triples (2) and (3) having datatype properties of the entities `dbr:Barack_Obama` and `dbr:Calvin_Coolidge` taken from DBpedia. The dotted boxes show the types for the two literals that we intend to compute, supplementing the syntactic type `xsd:string` which is already available. Whenever a semantic type can be computed for a literal, it can be used in practical applications for inferencing, grouping, and matching. For example, both computed types are `rdfs:subClassOf dbo:Politician` class in the DBpedia ontology and hence can be utilized for matching or grouping of related literals.

Datatype properties may have been provided instead of object properties for entities in datasets for various reasons, such as (i) the creator was unable to find a suitable entity URI for the object value, and hence chose to use a literal instead, (ii) the creator of the triple did not want to attach more details to the value and hence represented it in plain text, (iii) the value contains only basic implementation types like integer, boolean, and date, and hence not meaningful to create an entity, or (iv) the value has a lengthy description spanning several sentences (e.g., `dbo:abstract` property in DBpedia) that covers a diverse set of entities and facts. We attempt to assign a semantic type by analyzing cases

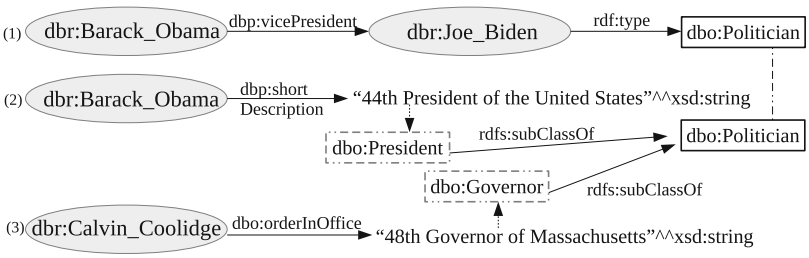

Fig. 1. Two triples corresponding to datatype properties and one triple corresponding to an object property taken from DBpedia. Computed types are shown in dashed boxes.

[2] http://www.w3.org/TR/owl-ref/#Property.

(i) and (ii) for text (up to a sentence long, delimited by a period) and avoid assigning a "type" to lengthy literal values as mentioned in (iv) because its focus is not clear (and multiple conflicting types can result).

3 Problem Statement

3.1 Typing Datatype Properties

Problem Statement: Let S P O be an RDF triple specifying subject (S), property (P), and object (O), and C be all classes (in the schema) for the set of triples D. If P is an object property, then O is an entity (i.e., individual). The type of O is a class assignment to O via the RDF triple, O `rdf:type` c, where $c \in C$. We refer to this as "semantic typing" in this work. Since the value of a datatype property instance is a literal, no semantic typing can be found for the value. We want to suggest a class $\bar{c} \in C$ for the value of a datatype property in addition to the syntactic type. As explained earlier in Sect. 2, we focus on text that is up to one sentence long. This realistic restriction has been imposed to ensure entity and type coherence. That is, deriving a unique type, which may not apply for longer texts, say a paragraph. Figure 1 illustrates types suggested for the object values of the triples (2) and (3).

For clarity of presentation, we define Type Set ($TS(v)$) for property value v as the set of classes that are assigned (via `rdf:type`) or inferred (via `rdfs:subClass Of`) from the class set C. If p is an object property, then $|TS(v))| > 0$, otherwise $|TS(v)| = 0$.

3.2 Ranking and Grouping Datatype Properties

Preliminaries: Let E, P, and L be sets of all entities, properties, and literals, respectively, in the triple set D (capturing a knowledge graph or dataset). V is the set of all property values and $V \subseteq E \cup L$. E and P are represented by URIs. An entity $e \in E$ is described using property-value pairs of the form (p, v) where $p \in P$ and value $v \in V$. Each of these property-value pairs of an entity is called a *feature* f of the entity. Furthermore, an entity and one of its features together correspond to an RDF triple in D. $prop(f)$ and $val(f)$ are two functions that return the property and its object value, respectively, for a feature f. The *feature set* of an entity e, denoted by $FS(e)$, is the set of all features that can be found for e in D. Using these notions, a *faceted entity summary* of entity e, which is a subset of all features that can be associated with e, is defined as follows (reproduced from [6] for completeness).

Definition 1 ***(Facet).*** *Given an entity e, a set of facets F(e) of e is a partition of the feature set FS(e). That is,* $F(e) = \{A_1, A_2, \ldots A_n\}$ *such that F(e) satisfies the following criteria: (i) Non-empty:* $\emptyset \notin F(e)$. *(ii) Collectively exhaustive:* $\bigcup_{A \in F(e)} A = FS(e)$. *(iii) Mutually (pairwise) disjoint: if* $A_i, A_j \in F(e)$ *and* $A_i \neq A_j$ *then* $A_i \cap A_j = \emptyset$. *Each* A_i *is called a facet of e.*

Faceted entity summary computation requires the identification of facets. Note that there can be multiple partitions for a given feature set, but in faceted entity summary generation, we compute a unique desirable partition by grouping related features together using a clustering algorithm that employs property name expansion (using WordNet[3]) and the "type information" associated with property values [6]. Informally, if the number of facets is n and the size of the summary is k, *at least one feature* from each facet is included in the summary when $k > n$. This can be achieved by iteratively picking features from each facet based on their ranking. If $k \leq n$, then *at most one feature* from each facet is included in the summary to improve diversity. Definition 2 formalizes the aforementioned idea about a faceted entity summary in set-theoretic terms.

Definition 2 ***(Faceted Entity Summary).*** *Given an entity e and a positive integer $k < |FS(e)|$,* faceted entity summary of e *of size k, $FSumm(e,k)$, is a collection of features such that $FSumm(e,k) \subset FS(e), |FSumm(e,k)| = k$. Further, either (i) $k > |F(e)|$ and $\forall X \in F(e)$, $X \cap FSumm(e,k) \neq \emptyset$ or (ii) $k \leq |F(e)|$ and $\forall X \in F(e), |X \cap FSumm(e,k)| \leq 1$ holds, where $F(e)$ is a set of facets of $FS(e)$.*

The ranking of features in each facet is computed using the informativeness $Inf(f)$ of feature f (Eq. 1) and popularity $Po(v)$ of value v, where $v = val(f)$ (Eq. 2). $N = |E|$ is the total number of entities in triple set D. The final rank score $Rank(f)$ is computed using Eq. 3. In our prior work [6], these ranking features have only been defined for object properties. In Sect. 4.2, we discuss how to adapt them for datatype properties.

$$Inf(f) = log(\frac{N}{|\{e \in E | f \in FS(e)\}|}) \tag{1}$$

$$Po(v) = log|\{triple\ t \in D | \exists\ e, f : t \equiv (e\ prop(f)\ v)\}| \tag{2}$$

$$Rank(f) = Inf(f)\ *\ Po(val(f)) \tag{3}$$

Problem Statement: Ranking and grouping features of object properties can be achieved using the entities that their values represent. For ranking, we can utilize Eqs. 1 to 3, and for grouping semantically related features, we could use their types [6]. Ranking features belonging to datatype properties cannot be done similarly because their literal values do not have a unique representation across the dataset (which URIs do provide for object properties) as the same entity may be referred to using minor variants of a literal. Therefore, we need to reflect related entities in ranking datatype properties by modifying Eqs. 1 to 3. For example, consider the third triple of Fig. 1 where we can spot: `dbr:Governor` and `dbr:Massachusetts`; here, we can use their frequency as opposed to checking the frequency of the entire literal value of the property.

[3] https://wordnet.princeton.edu/.

Grouping (conceptually similar) datatype properties is non-trivial compared to grouping object properties where types are available. Note that multiple entities and/or classes spotted in a datatype property value can confuse the groupings. For example, the second triple in Fig. 1 has the entity `dbr:United States` having the type `dbo:Country` but eventually results in the class `dbo: President` as the type. In general, this requires recognizing multiple types (e.g., country and president) and then resolving them suitably (e.g., to president) to enable the grouping of similar triples (among both object and datatype properties). See triple (1) and (2) in Fig. 1, where the first represents an object property and the second represents a datatype property. In fact, both values convey information about a person, while for the datatype property value, it is not explicit. The object property clearly has a type assigned to its value and if we compute the type for the datatype property as `dbo:President`, then we can abstract their values to type `dbo:Politician` which can be inferred for the datatype property value using `rdfs:subClassOf`.

4 Approach

First, we will investigate how to compute types for the values of datatype properties and then utilize them in grouping related properties and values based on the computed types. We will also discuss how to generate entity summaries based on new ranking measures for the datatype properties and groupings.

4.1 Typing Datatype Property Values

Determining the relevant type for a datatype property value is challenging due to several reasons. First, picking some term used in the literal value to determine the entity or class for the datatype property value does not work. For example, in triple (3) in Fig. 1, if we select the term "Massachusetts" as the entity to represent the entire value and use its type `dbo:PopulatedPlace` as the type, we obtain an incorrect interpretation. The main focus of the text is the term "Governor" and not "Massachusetts", as it conveys information about the governor. Therefore, we propose identifying this term, which we call the *focus term*, by analyzing the grammatical structure of the text. Then, we match the identified focus term to a suitable entity or a class in deriving the type for the value.

We utilize the Collins head word detection technique [2] to identify the focus terms. The Stanford CoreNLP[4] API offers an implementation of this technique using parse trees. We use the UMBC semantic similarity service [9] to compare the suggested class and the set of given classes. It facilitates the computation of phrase similarity, which generalizes and improves upon WordNet-based similarity. The algorithm for generating a type set ($TS(v)$) for a datatype property value v is presented in Algorithm 1.

Algorithm 1 shows details of the method *getTypesForText*, which generates types for the input text (datatype property value). We avoid processing if the

[4] http://nlp.stanford.edu/software/corenlp.shtml.

Algorithm 1. getTypesForText($Text\ v$)

```
1: initialize Set types to {} and pre-determined Integer n
2: Set X ← getPhrases(v)
3: for each Phrase x ∈ X do
4:    if isNumeric(x) then
5:       Set cls ← predefined date/numeric type
6:    else
7:       Set ngrams ← getNGrams(x, n)
8:       Text focusTerm ← parseHeadWord(x) {head word identifier}
9:       Set cls ← getTypeFromLabel(focusTerm)
10:      if isEmpty(cls) then
11:         cls ← getTypesFromNGrams(focusTerm, ngrams)
12:      end if
13:      if isEmpty(cls) then
14:         cls ← getMatchedType(focusTerm) {semantic matching}
15:      end if
16:   end if
17:   types ← cls
18: end for
19: return types
```

input text is more than one sentence long (segmented by "period"). If the identified sentence has phrases delimited by comma, we segment them (lines 2–3) and generate types for them. This is because these phrases normally align for the same abstract meaning (e.g., "Austrian-American bodybuilder, actor", "Denison, Texas"). We identify numeric or date values using simple regular expressions (lines 4–5). If the value is not numeric, we start the type computation process for the phrase by identifying n-grams associated with the phrase up to the maximum token length of n (line 7). Then, we retrieve the focus term by parsing the phrase using the head word identifier (line 8). Next, we check whether there is an exact match of the focus term and any of the types (via `rdfs:label` of classes) in the dataset. If a match is found, we take the class as the type of the phrase (line 9). Otherwise, we further analyze all the generated n-grams with the focus term to infer a type in the *getTypesFromNGrams* method (line 11). If there is still no match, we compute the similarity scores of the focus term against all the types in the dataset (via `rdfs:label` of classes) and get the highest match (>0) as the type of the phrase (line 14). Finally, we aggregate types generated for each phrase to obtain the set of types for the input text.

The method *getTypesFromNGrams* processes the n-grams set to allow for a maximal match of entity labels. It processes n-grams to extract types only if they contain the focus term. For each of those n-grams that contain the focus term, we check to see whether there is an exact match of the n-gram to a type. If no match is found, we spot entities for the n-gram and then get the types of those entities. We spot entities by exact matching of their labels (`rdf:label`) to n-grams. Looking for n-grams that contain the focus term (in descending order of n-gram token lengths from n to 1) can improve the quality of the identified types. For example, consider "Harvard Law School" as a datatype property value in DBpedia. The identified focus term for this phrase is "School." When we start processing n-grams in descending order of n, we encounter "Harvard Law School" as the first candidate for typing. This matches the entity `dbr:Harvard_Law_School` whose

type `dbo:Educational_Institution` is then taken as the type of the phrase. We do not generate types for long text[5] (e.g., paragraphs) because they cannot be unambiguously typed as they can represent many different entities (with contrasting descriptions) and need further analysis to pick the correct type.

4.2 Grouping and Ranking Datatype Property Features for Summaries

The approach to rank and group object properties to create faceted entity summaries is presented in [6]. The idea of generating faceted summaries focuses on grouping related features (i.e., property-value pairs) and then picking the highest ranked ones from each group. Next, we discuss how to group and then rank datatype property based features.

Grouping Datatype Property Features: Grouping of features can be done at two levels: exact/ syntactic similarity and semantic/abstract similarity. Exact/ syntactic similarities can result in very fine-grained groups, while we are interested in groups based on their abstractions in FACES [6]. For example, triples (2) and (3) in Fig. 1 present two literal values that do not share any common token (no/less syntactic similarity). However, when we compute a type for each, they are sub-types of the class `dbo:Politician`. The clustering algorithm that uses such type information of the values as in FACES can group them together. Further, it can also group features of both object and datatype properties which was hitherto not possible. For example, It can group features represented by triples (1) and (2) in Fig. 1 because the values are indirect instances of the type `dbo:Politician`. Figure 2 illustrates grouping of similar features with same color using the clustering algorithm presented in [6].

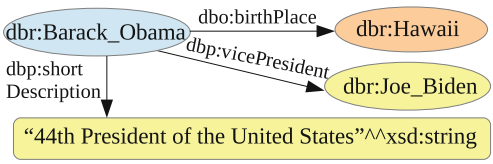

Fig. 2. Grouping both property features.

Ranking Datatype Property Features: We discuss ranking measures for datatype properties usable in the context of entity summarization. Recall that Eqs. 1 to 3 can be used only to rank object properties [6]. If we compute $Inf(f)$ as in Eq. 1 for datatype properties, it will have an artificially high value because the exact literal denoting an entity appears infrequently compared to URI references of the entities for object properties. As a consequence, every datatype property will have a high ranking score. To fix this discrepancy, we spot entities in datatype property values and get the frequency of entity URIs as a measure of informativeness and popularity. We first spot entities in the datatype property values by analyzing the n-grams generated in Algorithm 1. Let $ES(v)$ be the

[5] Note that we can still run the algorithm for each sentence to generate types.

set of all entities that can be spotted for value $v = val(f)$. We process all the n-grams generated for v (for a pre-defined n-gram token length n) and match them against the entity labels (`rdfs:label`) in the dataset to obtain $ES(v)$. Then, we choose the most popular (frequent) entity in the dataset from this set as the representative entity for v. The intuition is that humans spot and identify popular entities in text phrases and, hence, they can provide identifiable facts in the summary for datatype properties. Let *max(ES(v))* be a function that returns the most popular entity e_{max} from the set $ES(v)$ based on the frequency of appearance of each entity (using Eq. 2). For example, for triple (2) in Fig. 1, function *max* identifies `dbr:President` and `dbr:United_States` as the entities and picks the latter to be the most popular entity. Hence, it is used to calculate the informativeness of the feature and popularity of the phrase.

We compute the informativeness of a feature f of a datatype property, $Inf(f)'$, using Eq. 4. We check for occurrences of features in entities "similar" to f (as opposed to checking the same feature as in object properties) by matching datatype property names and values that contain the most popular entity e_{max}. Then, we count those entities to compute informativeness of the feature. That is, informativeness is inversely proportional to the number of entities that are associated with overlapping values containing e_{max}. N is the total number of entities.

$$Inf(f)' = log\Big(\frac{N}{|\{\begin{array}{r} e \in E | \exists\ f' \in FS(e) : prop(f) = prop(f')\ and \\ max(ES(val(f))) \in ES(val(f')) \end{array}\}|}\Big) \tag{4}$$

Similarly, for measuring the popularity $Po(v)'$ of a datatype property value v, we take the frequency of the most popular entity $e_{max} = max(ES(v))$ in v, as specified in Eq. 5. Then, the ranking score of feature f that belongs to datatype properties, $Rank(f)'$, is calculated using Eq. 6. When $ES(v) = \emptyset$, we take the denominator as the number of property instances in Eq. 4 and $Po(v)' = 1$ in Eq. 5, effectively ranking them low.

$$Po(v)' = log|\{\begin{array}{r} triple\ t \in D | \exists\ e, f : t \equiv (e\ prop(f)\ e_{max}) and \\ e_{max} = max(ES(v)) \end{array}\}| \tag{5}$$

$$Rank(f)' = Inf(f)' \ * \ Po(val(f))' \tag{6}$$

Faceted Entity Summaries Using Object and Datatype Properties: Eqs. 1–6 are used to rank features within each facet (cluster partition). We further extend the FACES approach by ranking facets using the average of feature ranking scores ($FacetRank(F(e))$) generated by Eqs. 3 and 6, as shown in Eq. 7 for a facet $F(e)$. $R(f)$ is a function that selects the proper ranking method depending on whether $prop(f)$ is an object or datatype property. Then, facets are ordered from the highest to the lowest FacetRank score and we iterate over them in that order to pick individual features for the summary.

$$R(f) = \begin{cases} Rank(f), & \text{if } prop(f) \textit{ is an } \text{object property} \\ Rank(f)', & \text{otherwise} \end{cases}$$

$$FacetRank(F(e)) = \frac{\sum_{f \in F(e)} R(f)}{n},\ where\ n = |F(e)| \tag{7}$$

Given the feature set $FS(e)$ of an entity e and a positive integer $k < |FS(e)|$, the adapted process for the faceted entity summary creation is as follows. (1) The feature set $FS(e)$ is partitioned into facets. The algorithm yields a dendrogram (hierarchical tree) for $FS(e)$ and it is cut at an empirically determined level to get the facet set $F(e)$ of $FS(e)$. (2) Features in each facet are ranked using the ranking algorithms (Eqs. 3 and 6). (3) Then the feature ranking scores of features in each facet are aggregated and averaged to get the facet ranking score (Eq. 7). (4) The top ranked features, from highest to lowest ranked facet, are picked (Definition 2) to form the faceted entity summary of length k.

5 Evaluation and Results Discussion

We evaluate our contributions in two steps: (1) evaluation of types generated for literal values of datatype properties and (2) evaluation of faceted entity summarization using features belonging to both types of properties. We empirically set $n = 3$ (n-grams) for both evaluations. More details on approach, evaluation, and examples are available at http://wiki.knoesis.org/index.php/FACES.

5.1 Evaluating Datatype Property Types

Generating types for all the available datatype property values is not meaningful because there are labeling properties that simply represent human readable names for entities. The RDFS (RDF Schema) standard defines the `rdfs:label` property to provide such information, but in practice, there exist many such labeling properties (e.g., foaf:name). Ell et al. [3] studied the characteristics of these properties by manually inspecting the properties and their instance data. Similarly, we created a list of labeling properties for our data sample and filtered them out. We extracted a sample of unique datatype property-value pairs from DBpedia (version 3.9 and 2015-04). *Precision* for the identified types of a property value v is defined in terms of $TS(v)$ in Eq. 8. Then, we define the *Mean Precision* (MP) of property values as in Eq. 9, where n is the number of property values in the sample that have $|TS(v)| > 0$.

$$Precision(TS(v)) = \frac{\#correct\ types\ in\ TS(v)}{|TS(v)|} \tag{8}$$

$$MeanPrecision = \frac{\sum_{i=1}^{n} Precision(TS(val(f_i)))}{n} \tag{9}$$

Mean Precision is the average of precision over the property values in the feature sample. When the MP value is higher, the algorithm generates many correct types over different property values. It is important to know how often the algorithm can generate at least one correct type. Therefore, we define *Any Mean Precision* (AMP) as in Eq. 10, where n is the number of property values

Table 1. Type generation evaluation.

	Mean precision (MP)	Any mean precision (AMP)	Coverage
Our approach	0.8290	0.8829	0.8529
Baseline	0.4867	0.5825	0.5533

in the sample that have $|TS(v)| > 0$. It computes the average of all the ceiling values of Precision($TS(v)$). If the algorithm generates at least one correct type for a value, it counts the precision as 1 in averaging. When AMP is higher, the algorithm generates at least one correct type often. When both the MP and AMP values are higher, the algorithm can be considered reliable.

$$AnyMeanPrecision = \frac{\sum_{i=1}^{n} \lceil Precision(TS(val(f_i)))\rceil}{n} \tag{10}$$

Table 1 shows the evaluation results performed by one evaluator. We constructed a baseline using a state-of-the-art tool to identify entities in the values and retrieved their types and super types (except `owl:Thing`). Specifically, we used DBpedia Spotlight [11] for this purpose and configured it with default parameters including the confidence of 0.5. We had a total of 1,117 unique property-value pairs after filtering out 118 labeling and noisy pairs. Coverage is the fraction of features that had a type generated. Our approach performed better compared to the baseline because we identify types using a combination of focus terms and matching entities and types. We did not measure recall because it is hard to produce an exhaustive list of all correct types for each value.

5.2 Evaluating Faceted Entity Summarization Use Case

We evaluated the proposed extended faceted entity summarization approach FACES-E against another state-of-the-art algorithm called RELIN [1]. It has been shown before that FACES outperformed RELIN for object properties [6]. RELIN has been the only tool to generate entity summaries for both datatype and object properties. We evaluate FACES-E against RELIN for the full range of features and show the benefits of the datatype property typing which enabled FACES-E to group features belonging to object and datatype properties in the partition algorithm. We randomly selected 20 entities from the FACES evaluation (DBpedia 3.9) [6] and another random sample of 60 entities from DBpedia version 2015-04 for a total of 80 unique entities. We retrieved object properties as earlier [6] and added datatype properties to each entity, filtering labeling properties (including date and numeric as in Sect. 5.1). We created a new gold standard for the entity samples by asking 17 human users to create summaries of length 5 and 10 for each of the 80 entities as the "ideal summaries" (total of 900 user-generated ideal summaries for both summary lengths). Each entity received at least 4 different ideal summaries and this comprises the gold standard. We use the same evaluation metrics as in [1,6]. When there are n ideal

Table 2. Evaluation of the summary quality (average for 80 entities) and %↑ = 100 * (FACES-E avg. quality - Other system avg. quality) / (Other system avg. quality) for $k = 5$ and $k = 10$, where k is the summary length.

System	k = 5		k = 10	
	Avg. Quality	%↑	Avg. Quality	%↑
FACES-E	1.5308	–	4.5320	–
RELIN	0.9611	59 %	3.0988	46 %
RELINM	1.0251	49 %	3.6514	24 %
Avg. Agreement	2.1168		5.4363	

summaries denoted by $Summ_i^I(e)$ for $i = 1, .., n$ and an automatically generated summary denoted by $Summ(e)$ for entity e, the agreement on ideal summaries is measured by Eq. 11 and the quality of the automatically generated summary is measured by Eq. 12. In other words, the quality of an entity summary is its average overlap with the ideal summaries for the entity in the gold standard.

$$Agreement = \frac{2}{n(n-1)} \sum_{i=1}^{n} \sum_{j=i+1}^{n} |Summ_i^I(e) \cap Summ_j^I(e)| \tag{11}$$

$$Quality(Summ(e)) = \frac{1}{n} \sum_{i=1}^{n} |Summ(e) \ \cap \ Summ_i^I(e)| \tag{12}$$

We used previously determined thresholds for both FACES-E and RELIN. RELINM is the modified version of RELIN where it discards duplicate properties in the summary [6]. For FACES-E, we cut cluster hierarchies at level 3 and set the cut-off threshold of the clustering algorithm (Cobweb) to 5. For RELIN and RELINM, we set the jump probability to 0.85 and the number of iterations to 10. The evaluation results of entity summarization are presented in Table 2. Note that the *summary quality* is better when it is closer to the *agreement* value. Agreement is low for this evaluation, as was the case with previous evaluations [6], because the number of features per entity was relatively high (on average 44 features per entity). Our intuition for ranking datatype property features by giving precedence to popular entity mentions in the value, grouping features based on types, including the generated semantic types for datatype property values, and ranking facets to select faceted summaries, have been validated by the high summary quality in this evaluation. We conducted a *paired t-test* to confirm the significance of the FACES-E's mean summary quality improvements over RELINM. For $k = 5$ and $k = 10$, p_values for FACES against RELINM are 8.24E-11 and 9.42E-12. When p_values are less than 0.05, the results are statistically significant. According to results shown in Table 2 and the paired t-test, FACES-E performed better and benefited from datatype property typing.

6 General Discussion and Future Directions

Our typing algorithm enabled FACES-E to process both types of properties, a limitation of FACES, improving coverage. Likewise, typing for datatype properties can facilitate a wide range of data processing applications. Property alignment [7] can be one such instance where similarly typed property values can be used to limit the properties analyzed for equivalence and relatedness.

In the first evaluation, our algorithm showed MP and AMP values of 0.82 and 0.88 compared to 0.48 and 0.58, respectively of the baseline. This aligns with our claim that typing for values needs to identify the correct and appropriate focus. Furthermore, our algorithm showed good coverage. Getting both MP and AMP values to a relatively high level is desirable for faceted entity summaries. This is because when the algorithm can generate a correct type most of the time, also with high precision, it helps to group semantically similar features together. This, in turn, facilitates the creation of high quality, "diversified" entity summaries evidenced by the second evaluation in Sect. 5.2. However, we also note that datatype properties in our entity sample have labeling and noisy properties (due to incorrect or missing details) which is normal for real-world datasets on the Web. We manually filtered such labeling properties; however, this can be a challenging and important problem to solve in the future.

It is possible to use the meaning of the property names and word sequence relationships of values for type generation. For this, a machine learning model similar to Conditional Random Fields (CRF) could be utilized whereas now, focus term detection drives the type computation. This can facilitate predicting whether a value can be typed or not and filtering out noisy and labeling property values. The absence of matching entities in DBpedia can stymie the generation of type information. These are common dataset quality and completeness issues orthogonal to our problem. For missing information, we can: (1) use type inferencing approaches [13] to generate missing types and (2) use a comprehensive set of ontology classes and entities (e.g., from LOD) in our approach.

Our approach does not generate "semantic types" for numeric and date value properties, and challenges exist for measuring their popularity in the dataset for ranking. These properties will be investigated in the future for grouping and ranking in faceted entity summary generation. Furthermore, a formal model/approach needs to be adapted for RDF semantics to encapsulate type generations for datatype properties and then they can be encoded in the datasets similar to object properties whereas now, we keep computed values separate.

7 Related Work

Type assignment to text fragments is known as Named Entity Recognition (NER) [12]. NER consists of two subtasks: segmenting and classifying segmented text blocks into pre-defined categories (types). Entity Linking (EL) [8] maps

entity mentions in text to their corresponding entities in knowledge bases. NER produces types for segments of the input text whereas EL identifies entities from which types can be inferred. NER and EL, however, differ from our problem in that they do not try to suggest a type based on the focus of the text but rather try to determine the types of all the entities present, causing ambiguity from the perspective of our problem (e.g., our evaluation with DBpedia Spotlight [11]).

Finding missing types for entities [13,14] is important for the reliability of datasets and reasoning. Paulheim and Bizer [13] infer types for entities in DBpedia and Sleeman and Finin [14] predict types of entities for efficient co-reference resolution. TRank [17] ranks entity types based on the context in which they appear (disambiguation). Fang et al. [4] use type information for search, and Tylenda et al. [18] generate summaries by analyzing type graphs. These approaches work on entities and/or object properties or infer types for entities, whereas we focus on typing datatype properties where no semantic types are available.

FACES [6] proposed a concise, comprehensive, and diversified approach for entity summarization for object properties and showed superior results over the state-of-the-art techniques and methods [1,15,16]. SUMMARUM [16] employs a PageRank based algorithm to rank and compute summaries for only object properties whereas RELIN [1] utilizes PageRank to compute summaries for both object and datatype properties. We developed FACES-E to utilize both object and datatype properties and showed superior results compared to RELIN.

8 Conclusion

We have investigated the problem of computing types for datatype property values. We generate types for a property value by: (1) exact and semantic matching of the focus term to class labels, and (2) spotting entities related to the focus term and retrieving their types. Our contributions in this work span over two significant problems: (1) enhancing datatype property values with type metadata, and (2) proposing FACES-E, which extends FACES to generate more comprehensive entity summaries using both types of properties. We evaluated both type generation and the extended entity summarization approach using the DBpedia encyclopedic dataset on the Web and showed improvement over the state-of-the-art. Our novel typing algorithm for datatype property values enhances data with additional semantics and, hence, is useful in applications beyond entity summarization, such as property alignment, data integration, and dataset profiling.

Acknowledgments. We acknowledge partial support from the National Science Foundation (NSF) award: EAR 1520870: Hazards SEES: Social and Physical Sensing Enabled Decision Support for Disaster Management and Response. Any opinions, findings, and conclusions/recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

References

1. Cheng, G., Tran, T., Qu, Y.: RELIN: relatedness and informativeness-based centrality for entity summarization. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 114–129. Springer, Heidelberg (2011)
2. Collins, M.: Head-driven statistical models for natural language parsing. Comput. Linguist. **29**(4), 589–637 (2003)
3. Ell, B., Vrandečić, D., Simperl, E.: Labels in the web of data. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 162–176. Springer, Heidelberg (2011)
4. Fang, Y., Si, L., Somasundaram, N., Al-Ansari, S., Yu, Z., Xian, Y.: Purdue at TREC 2010 entity track: a probabilistic framework for matching types between candidate and target entities. In: TREC (2010)
5. Gunaratna, K., Lalithsena, S., Sheth, A.: Alignment and dataset identification of linked data in semantic web. Wiley Interdisc. Rev.: Data Min. Knowl. Disc. **4**(2), 139–151 (2014)
6. Gunaratna, K., Thirunarayan, K., Sheth, A.: FACES: Diversity-aware entity summarization using incremental hierarchical conceptual clustering. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, pp. 116–123. AAAI (2015). http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9562/9233
7. Gunaratna, K., Thirunarayan, K., Jain, P., Sheth, A., Wijeratne, S.: A statistical and schema independent approach to identify equivalent properties on linked data. In: Proceedings of the 9th International Conference on Semantic Systems, pp. 33–40. ACM (2013)
8. Hachey, B., Radford, W., Nothman, J., Honnibal, M., Curran, J.R.: Evaluating entity linking with wikipedia. Artif. Intell. **194**, 130–150 (2013)
9. Han, L., Kashyap, A., Finin, T., Mayfield, J., Weese, J.: UMBC ebiquity-core: semantic textual similarity systems. In: Proceedings of the Second Joint Conference on Lexical and Computational Semantics, vol. 1, pp. 44–52 (2013)
10. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., et al.: Dbpedia-a large-scale, multilingual knowledge base extracted from wikipedia. Seman. Web J. **5**, 1–29 (2014)
11. Mendes, P.N., Jakob, M., García-Silva, A., Bizer, C.: DBpedia spotlight: shedding light on the web of documents. In: Proceedings of the 7th International Conference on Semantic Systems, pp. 1–8. ACM (2011)
12. Nadeau, D., Sekine, S.: A survey of named entity recognition and classification. Lingvisticae Investigationes **30**(1), 3–26 (2007)
13. Paulheim, H., Bizer, C.: Type inference on noisy RDF data. In: Alani, H., et al. (eds.) ISWC 2013, Part I. LNCS, vol. 8218, pp. 510–525. Springer, Heidelberg (2013)
14. Sleeman, J., Finin, T.: Type prediction for efficient coreference resolution in heterogeneous semantic graphs. In: 2013 IEEE Seventh International Conference on Semantic Computing (ICSC), pp. 78–85. IEEE (2013)
15. Thalhammer, A., Knuth, M., Sack, H.: Evaluating entity summarization using a game-based ground truth. In: Cudré-Mauroux, P., et al. (eds.) ISWC 2012, Part II. LNCS, vol. 7650, pp. 350–361. Springer, Heidelberg (2012)

16. Thalhammer, A., Rettinger, A.: Browsing DBpedia entities with summaries. In: Presutti, V., Blomqvist, E., Troncy, R., Sack, H., Papadakis, I., Tordai, A. (eds.) ESWC Satellite Events 2014. LNCS, vol. 8798, pp. 511–515. Springer, Heidelberg (2014)
17. Tonon, A., Catasta, M., Demartini, G., Cudré-Mauroux, P., Aberer, K.: TRank: ranking entity types using the web of data. In: Alani, H., et al. (eds.) ISWC 2013, Part I. LNCS, vol. 8218, pp. 640–656. Springer, Heidelberg (2013)
18. Tylenda, T., Sozio, M., Weikum, G.: Einstein: physicist or vegetarian? summarizing semantic type graphs for knowledge discovery. In: Proceedings of the 20th International Conference Companion on World Wide Web, pp. 273–276. ACM (2011)