Fuzzy Semantic Labeling of Semi-structured Numerical Datasets

Ahmad Alobaid[0000−0001−8637−6313] and Oscar Corcho[0000−0002−9260−0753]

Ontology Engineering Group, Universidad Politécnica de Madrid,
28660 Boadilla del Monte, Madrid, Spain,
{aalobaid,ocorcho}@fi.upm.es

Abstract. SPARQL endpoints provide access to rich sources of data (e.g. knowledge graphs), which can be used to classify other less structured datasets (e.g. CSV files or HTML tables on the Web). We propose an approach to suggest *types* for the numerical columns of a collection of input files available as CSVs. Our approach is based on the application of the *fuzzy c-means* clustering technique to numerical data in the input files, using existing SPARQL endpoints to generate training datasets. Our approach has three major advantages: it works directly with live knowledge graphs, it does not require knowledge-graph profiling beforehand, and it avoids tedious and costly manual training to match values with *types*. We evaluate our approach against manually annotated datasets. The results show that the proposed approach classifies most of the types correctly for our test sets.

Keywords: Fuzzy Clustering, Semantic Labeling, Semantic Web

1 Introduction

A massive number of data are stored and made publicly available on the Web in semi-structured formats, such as spreadsheets. This is especially the case for open data made available by public administrations, since the publication of CSV data grants them three stars in the 5-star open data scheme[1].

A major drawback of the publication of data in spreadsheets is the difficulty for potential data consumers to understand and interpret their content. This is because the terms used for column headings in these files are commonly not sufficiently informative and lack a data dictionary where their meaning is provided. Therefore, the automatic classification of such semi-structured data sources may be useful to improve their usage. For example, such characterization may allow search engines to improve the relevancy of results [4]. It may also be used to (partially) automate the generation of mappings (e.g. RML [7] and R2RML [6]) that may be used to generate RDF on the fly without actually transforming the data.

Meanwhile, data are also exposed on the Web by means of Linked Data principles or via SPARQL endpoints, which can be considered as rich sources

[1] http://5stardata.info/en/

The final authenticated version is available online at https://doi.org/10.1007/978-3-030-03667-6_2.

of more structured and well-described data. Our hypothesis is that such data can be useful to train models that are able to characterize the numerical semi-structured data sources that we were referring to in the previous paragraph.

In this paper, we describe an approach for the characterization of semi-structured data sources (e.g., CSVs) that uses the content available in SPARQL endpoints for such characterization. Our approach is based on the usage of the *fuzzy c-means* clustering technique. We have identified the following advantages:

- It is domain agnostic. That is, it performs the semantic labeling of semi-structured data sources regardless of their domain, what makes it applicable to a wide range of datasets.
- No manual training is needed. It does not require users to manually type samples of the data beforehand or to use a training dataset that has been constructed before. Instead, it works with existing data available as SPARQL endpoints.
- It does not require exact matches for the numerical values whose columns it classifies. The correct *typing* of data sources is not prevented by having values that are updated over time (e.g. max temperature observed in a city) or entities/values that are not shared between training and testing data (e.g. heights of a local sports team that are not in the knowledge graph).
- Works with live knowledge graphs. It does not require the knowledge graph to be downloaded locally (which is not always feasible).
- No knowledge-graph profiling is required. It works directly with the data and does not need the knowledge graph to be profiled before being able to label the input data sources.

Several approaches have been proposed in the state of the art for the purpose of classifying (a.k.a. semantic labeling) semi-structured data. They use a range of techniques (e.g. graphical probabilistic models [8,9,18,21], linear regression [4,12], decision trees [17], etc.). These are described in the following section, where we discuss their advantages and disadvantages when compared to the approach presented in this paper.

Section 3 describes our approach, based on fuzzy clustering, to classify numerical data sources, with a running example coming from the domain of Olympic Games. In section 4, we describe the details of the data collection process from SPARQL endpoints, which we use to obtain data for the classification task. In section 5 we evaluate our approach and discuss the results obtained in our experiments. Finally, section 6 includes a reference to our planned future work.

2 Related Work

Different approaches have been used in the literature to perform semantic labeling. We understand by semantic labeling the process of assigning *types* from knowledge bases to values or a collection of values. In this section, we describe some of the most relevant approaches that have been proposed in the past for tackling semantic labeling of tabular data sources.

Cafarella et al. [4] present an approach for semantic matching of Web tables extracted from Web pages (from the Google crawl). Their approach exploits an attribute correlation statistics database (ACSDb) that contains the frequency of occurrences of schemas and attributes to compute the occurrence probabilities of an attribute given another. For relations retrieval, they use schema ranking which is based on a linear regression estimator with different features such as the number of hits on the table header. They also use a schema coherency score based on Point-wise Mutual Information, which provides a sense on how strongly two items are related.

Limaye et al. [9] use probabilistic graphical models for semantic labeling, entity detection and relation extraction using YAGO. They use three random variables; column type, entity, and the relation between two columns, which are used to construct the features. The features are based on cosine similarity of the cell and column headers, compatibility of the entity and semantic types, and relationship compatibility between different columns types and entity pairs. They weight the features using a weight vector that learns using a generalized Support Vector Machine method.

Syed et al. [17] build an index from Wikipedia pages including the titles, redirects, first sentences, categories, and types. They discover ontology properties from Wikipedia articles and the class hierarchy is inferred from the slots and fillers. Slots are predicted for different entities and compared with the DBpedia infobox ontology and Freebase. Then, Wikitology is used to link entities to the right Wikipedia entities. In addition to that, the relation between pairs of values is discovered by querying DBpedia for pairs of values in each row and selecting the maximum appearing relation.

Ventis et al. [19] semantically label web tables using two databases: isA database and relation database. They start with the isA database to identify the class of each column. After identifying the classes of each column, they inspect the relation between two columns using the relation database, which is in the form of (a, R, b), where a is an instance of class A , b is an instance of class B, and R is the relation between a and b.

Goel et al. [8] semantically label source attributes using probabilistic graphical models, namely Conditional Random Fields, exploiting the latent (hidden) structure within the data. They tokenize the values and apply features depending on the token type. The features for alphabetic tokens are the token length, the starting letter, whether the token is capitalized, all upper case and the token value. For numeric tokens, the length of the number, the number of digits before the decimal points, the number of digits after the decimal points, whether the numeric token is negative, starting digits, the place of the unit and the tenth place unit digit. For the symbol token, the feature is only the value itself. Besides that, they consider the relationship between neighboring labels, tokens and attribute labels.

Zhang et al. [21] match and semantically annotate numeric time-varying attributes in web tables using collective inference based on belief propagation. They take into account tables' headers and context (e.g. surrounding text). They use

heuristics to find the subject column and split each table into (n-1) tables, the subject column with each of the other columns. They connect attributes if they have the same year and unit/scale. They also connect attributes if they have the same year and they match after applying any of the conversion rules (e.g. Euro = 1.3 * USD). Possible labels for each table are gathered from the header of the table and its context information and inconsistent labels are eliminated by the use of mutual exclusive labels.

Ritze et al. [15] present T2K, an iterative matching algorithm to match Web Tables to knowledge bases. This algorithm performs entity-level matching and schema-level matching. Entity column is picked by examining columns with the most distinct values and data types are detected using predefined regular expressions. Similarities are computed among values between the Web Tables and DBpedia. Matches between Web Tables and DBpedia properties are aggregated and classes that do not belong are eliminated.

Taheriyan et al. [18] build a semantic model that represents the relationship between fields in data sources rather than only annotating attributes as semantic types. It types the data sources semantically, and then use that semantic labeling with confidence intervals to construct the semantic model. Then, it builds a graph with links that correspond to candidate types inferred by the ontology.

Pham et al. [12] propose a semantic labeling approach based on logistic regression. The features they rely on are similarity measures using Jaccard similarity and TF-IDF besides the attribute name (in the header), Kolmogorov-Smirnov test and Mann-Whitney test. The importance of each feature is computed for each domain, and that importance weight (which depends on the training data) is used afterwards for classifying the data sources.

Neumaier et al. [11] notice that a semantic type can appear in different contexts, so they aim to create a context for the semantic labels instead of mapping properties only. They represent that as a tree with each one of the children being a context. After that, they build a hierarchical background knowledge graph. For constructing the background knowledge graph, they use the *rdfs:subClassOf* and property-object pairs. For predicting the new data sources, they use the Kolmogorov-Smirnov test and nearest neighbors over the background knowledge graph.

From this initial analysis of the state of the art in semantic labeling we obtain the following set of conclusions:

- Some approaches focus only on textual data [9,17] or numerical data [11,21]. Zhang et al. [21] focus on straight numerical matching or numerical matching after applying conversion rules.
- Some approaches use SPARQL endpoints as learning sources, such as YAGO in [9] and DBpedia in [11,15], what makes them applicable to changing learning sets. While others are more focused on learning from scraped web pages which do not provide such ease to focus on a specific domain[4,8,17,19].

```
SELECT distinct ?property ?class WHERE {
?property rdfs:domain ?class. ?property rdfs:range ?range.
FILTER(?range IN ( xsd:float, xsd:double, xsd:decimal,
xsd:int, xsd:nonPositiveInteger, xsd:negativeInteger, xsd:long,
xsd:integer, xsd:short, xsd:byte, xsd:nonNegativeInteger,
xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort,
xsd:unsignedByte, xsd:positiveInteger))}
```

Listing 1: query numerical properties and their corresponding classes using domain and range

– Despite the fact that these approaches may be automatic or semi-automatic[2], some of them require manual actions (e.g., provide predefined conversion rules [15,21], a blacklist of properties [11] to improve the accuracy and abbreviations resolution [15,21].

3 Approach

Our approach can be divided into three main steps. The first step is *data extraction* for model training, where we extract the data of interest from a chosen SPARQL endpoint. The next step is *training the model* using the extracted data from the previous step. The third step is the *classification (typing) of the input data* using the trained model. The last two steps are explained in section 3.2 as both are closely related to the fuzzy clustering technique.

3.1 Training-Data Extraction

The data extraction step looks for numerical properties and its values. It extracts them from a specified SPARQL endpoint. We explored three ways of getting numerical properties.

Extraction method 1: the use of *rdfs:domain* and *rdfs:range* to extract classes and properties with numerical objects. We query classes with properties' *range* matching any of the numerical data types [14] (see Listing 1). This approach is fast even with large knowledge bases like DBpedia. The problem is that the obtained class/property pairs cover only a small subset of the actual data, which concurs with the findings reported by Weise et al. [20].

Extraction method 2: We query the A-Box to get the numerical properties. The query (Listing 2) proposed by Neumaier et al. [11] to get all the numerical properties times out as, reported in their paper. We modified the query to fetch numerical properties for a single class (Listing 3), but that query was timing out as well.

[2] We are not referring here to the gold standards that are built manually or the semantic models that are constructed by domain experts.

```
SELECT ?p, COUNT(DISTINCT ?o) AS ?cnt
WHERE { ?s ?p ?o. FILTER (isNumeric(?o))} GROUP BY ?p
```

Listing 2: extract all numerical properties for all classes [11]

```
SELECT ?property WHERE {
?subject a dbo:SoccerPlayer.
?subject ?property ?val FILTER(isNumeric(?val))
} GROUP BY ?property
```

Listing 3: extract numerical properties for a given class

Extraction method 3: query all properties for a given class and then filter the numerical properties on the client-side (Listing **??**). After getting the list of properties, we query the endpoint to get the list of *objects* for each class/property combination. The added value of filtering numerical properties on the client side, besides overcoming the timeout problem, is distinguishing between numerical properties and properties that are not numerical but happen to have numerical values as a wrong entry (which happen often).

3.2 Fuzzy Clustering

Fuzzy c-means clustering is an unsupervised machine learning technique that generalizes *k-means* clustering [3]. In k-means, each data point belongs to a single cluster, while in fuzzy c-means each data point may belong to multiple clusters. The belonging of each data point is represented as a vector of values between 0 and 1, inclusively. This vector is often referred to as the *membership* vector. The values in each membership vector should sum to 1. The values in the membership vectors reflect how much it belongs to the corresponding cluster; the closer the value to 1 the stronger the belonging is to that cluster.

Notation and Variable Names: [3]

- m: weighting exponent to control fuzziness.
- d_{ik}: the distance between a datapoint k and a cluster center i.
- N: the number of data points.
- y_k: the data point value at index k.
- c: number of clusters.
- v_i: cluster at index i.
- u_{ik}: the membership value of a data point at index k to cluster i

[3] We use the same notation and variable names as in [3] (Bezdek et al.) 1984.

Learning: learning from the data extracted from the SPARQL endpoint, clusters are formed. Each cluster represents a class/property pair (e.g. <dbo: SoccerPlayer, dbo:height>). From the data extraction step, the number of clusters and their class/property pairs are fetched, but not the centers of clusters. In the learning step, the cluster centers (centroids) are the values of the computed features.

Features: there are two features used to calculate the centers of the clusters, the mean and the standard deviation. They are calculated for each cluster values (objects of a numerical property of a class) to become the center of it.

Clusters Centroids: computing the cluster centers (centroids) in fuzzy c-means requires the initial values for the membership matrix (which is composed of the membership vectors). We set the initial membership values to zero except for the class/property cluster they belong to, which would be 1. We apply Equation 1 which uses the initial membership matrix and the computed features to compute the center of each cluster [3].

$$v_i = \left(\sum_{k=1}^{N} (u_{ik})^m y_k \Big/ \sum_{k=1}^{N} (u_{ik})^m \right) ; 1 \leq i \leq c \tag{1}$$

Classification: using the centroids, semantic properties from the SPARQL endpoint are assigned to numerical columns in the input file. This is performed using the *fuzzy c-means* clustering technique. The features of each numerical column in the input file are computed. The set of features are fed to the model (which contains the clusters) and results in a membership matrix. The membership matrix is computed using Equation 2 by Bezdek et al. [3]. The membership matrix is composed of a list of membership vectors where each numerical column in the input file has a membership vector. Each membership vector contains the belonging to each of the corresponding clusters.

$$u_{ik} = \left(\sum_{j=1}^{c} \left(\frac{d_{ik}}{d_{jk}} \right)^{2/(m-1)} \right)^{-1} ; 1 \leq k \leq N; 1 \leq i \leq c \tag{2}$$

Overall approach: after extracting numerical properties and their values from the endpoint, the features for each class/property pair is computed for their values. These features will be the centers for their corresponding class/property cluster. A model is then created using the clusters (centers) and their corresponding class/property combinations. The membership matrix is initialized accordingly (with value 1 to the matching cluster and 0 elsewhere). The model is used next to classify each numerical column in the input file(s) using the

Equation 2. The classification of each column will result in a membership vector showing the fuzzy belonging for each column in the input file(s) to clusters computed from the endpoint (see the algorithm in Figure 4).

Looking at the algorithm in Listing 4, the **data extraction** function starts by getting classes and properties (lines 1-2). Then, the values for each class/property combination (lines 4-5) are extracted and stored if the values are numerical (lines 6-7). Then, the values and the class/property pairs are returned (line 10). The variable $list\_of\_values$ (line 3) is a list of columns where a column from the function $getValues$ is appended to the list in each iteration. Next, the **learning** function is called (line 31) with the list of values for each class/property combinations and the list of numerical class/property pairs as arguments. After that, the features for the list of values for each class/property pair is computed (line 15). The features are considered the center of the cluster. The cluster is then created containing the class/property pair and center (line 15) and appended to the list of clusters (line 16). The membership matrix is then calculated from the clusters centers and then returned with the list of clusters (lines 18-19). The last part of the algorithm is to **classify** the input files with computed clusters. It sends the clusters, the membership matrix, and the input files as arguments. The first step in the classify function is getting numerical columns from the input files (line 22). Each of the numeric columns is then classified in the form of membership vector and indexed by the file name and the index of the column (lines 24-27). The classification is returned (line 28).

Differences to classical fuzzy c-means: the above approach is not how classical fuzzy c-means clustering works exactly as in [3], despite the use of the same formulas for computing cluster centers and for classification. The first difference is in setting the initial membership matrix. In our case, we set it to 1's for the corresponding cluster, and 0's everywhere else. This is because we already know the cluster that each of the data points belongs to, which we already extracted from the endpoint. The second difference lies in the flow of the algorithm. In the classical version, the cluster centers adapt and change after the clustering of the data points which in return, affect the membership of the data points. This keeps looping until a threshold is met. In our case, clusters are computed once as we know the cluster each point belongs to (which we extract from the knowledge graph). Hence, the membership for the testing data is computed once. The last difference is that our approach resulted in a classification rather than a mere clustering. In other words, along with the membership result, we obtain the *type/class* for each cluster.

4 Evaluation

In this section, we evaluate our approach, checking how accurately it classifies the numerical columns of the input files. We compare the suggested types with our manual annotation and report the scores for the top 1, top 3 and top 5

```
                                                          ▷ Data Extraction
    function DATAEXTRACTION(classes)
2:      classes_properties ← getClassesAndProperties(classes)
3:      list_of_values ← ∅
4:      for all class, property in classes_properties do
5:          values ← getValues(class, property)
6:          if is_numeric(values) then
7:              list_of_values ← list_of_values + values
8:          end if
9:      end for
10:     return list_of_values, classes_properties
11: end function
                                                          ▷ Learning
12: function LEARNING(list_of_values, classes_properties)
13:     clusters ← ∅
14:     for all values, (class, property) in list_of_values, classes_properties do
15:         center ← compute_features(values)
16:         clusters ← clusters + new Cluster(class, property, center)
17:     end for
18:     membership_matrix ← membership_from_clusters(clusters)
19:     return clusters, membership_matrix
20: end function
                                                          ▷ Classifying
21: function CLASSIFY(clusters, membership_matrix, input_files)
22:     input_columns ← getNumericalColumns(input_files)
23:     classifications ← ∅
24:     for all file_name, column_no, column in input_columns do
25:         membership_vector ← predict(clusters, membership_matrix, column)
26:         classifications[file_name][column_no] ← membership_vector
27:     end for
28:     return classifications
29: end function
                                                          ▷ Overall Approach
30: list_of_values, classes_properties ← DATAEXTRACTION(classes)
31: clusters, membership_matrix ← LEARNING(list_of_values, classes_properties)
32: classifications ← CLASSIFY(clusters, membership_matrix, input_files)
```

Listing 4: Algorithm to label semi-structured data sources

candidate types for each property. We have the data and the source code publicly available [1,2].

Our evaluation is performed over two sets of data: Olympic Games and Web Data Commons [16]. The first one is collected by us and we report the process in Section 4.1. The Web Data Commons contains a set of web tables crawled from the web and annotated manually. In this section, we explain the testing for each used dataset and describe the datasets.

4.1 Olympic Games Dataset

Here we explain the process of collecting the Olympic Games data, describe the data, and report the results.

Data Gathering After choosing the domain, we went to the Wikipedia page about the Olympics of 2020 [4] and collected all the scheduled sports. Using Google search engine, we search for the data found in the SPARQL endpoint. We explore the returned links from the first two pages of results returned by Google. Data gathering is performed in a systematic way to reduce possible bias. In case multiple results are found, we take the first one that we come across. We focus on data that are in tabular forms such as CSV, TSV, Excel or web tables. Since we are using DBpedia for this test, we did not collect any data from Wikipedia (except for getting the list of games) to reduce the amount of possible bias as DBpedia has data extracted from Wikipedia. We show a description of the data set in Table 1.

Classes Lookup We use Loupe[5] to look for classes from DBpedia. Loupe is an online tool for inspecting datasets [10]. We ignore any property in case it exists in the SPARQL endpoint and doesn't exist (or couldn't be found) on the web or the other way around. Properties co-existing in both are compiled for the tests. It is important to note that our algorithm is not dependant on Loupe, it is only used in data gathering step for the experiment.

Table 1: Information about the Olymic Games dataset

Number of files	12
Number of classes (Concepts)	12
Number of numerical columns	24

Experiment and Results We perform the test for each class, so we create multiple models, one for each class. Then, for each model, we use the corresponding input files. For example, for the class dbo:Cyclist, we use a CSV file that contains information about cyclists and their weights. We do that for all the gathered classes. The results of each model are validated against the manual

[4] `https://en.wikipedia.org/wiki/2020_Summer_Olympics`
[5] `http://loupe.linkeddata.es/`

annotation. We compute the classification score of each class for k=1, k=3, and k=5, whether it is in the top k candidates. So for k=5, we check if the correct classification is in the top 5 candidates. After that, we average the scores for each k for each class. The classification scores are reported in Table 2. We refer to our approach as FCM (Fuzzy c-means). The score is the percentage of correct predicted properties in the top k candidates. For example, for k=1 the score is 0.96 while the probability of the correct property to be picked randomly is 0.0025. We compare our approach to the work of Neumaier et al. [11]. We tested it with the same collection of CSV files and report the results in Table 2 (referred to it as "MLSL" which stands for Multi-Level Semantic Labeling). We also explored having a model with all the classes. This would make it generally more challenging to classify as the number of clusters increase. We applied our algorithm and we show the results in Table 2, referred to as "Score (FCM) Merged". We discuss the results in Section 4.3.

Table 2: Classification score of English DBpedia's test

Top K-Candidates	Score (FCM)	Score (FCM) Merged	Score (MLSL)	Random
1	0.96	0.79	0.07	0.0025
3	0.96	0.92	0.07	0.0075
5	0.96	1.0	0.07	0.0125

4.2 Web Data Commons Dataset

The Web Data Commons (T2Dv2) contains a total of 237 annotated tables, with 319 numerical columns in total spanning 41 concepts[6]. In T2Dv2, properties are not annotated, only the entity columns are annotated with DBpedia classes. We utilized this information for building the models.

Experiment and Results We start by transforming the Web Tables from JSON to CSV. We create a model for each Web Table if it is not created and run the classification algorithm. A model will be created for each file using the class URI that each Web Table is annotated with in T2Dv2. We found that some of the Web Tables do not include numerical columns, so they will not have any classification. For each file, the corresponding model is used for the classification. We found 319 numerical columns in total, 232 of them can be understood while 87 of them were vague. Out of the columns that we understood, 137 of them actually existed in DBpedia. Our application was able to classify 124 out of the 137 columns (Table 3). We report the detailed results for each column (e.g. if it is vague, if it is found in DBpedia, etc.) with the rest of the results on the web [2].In Table 4, we show the scores. The score is the of number correct annotated columns divided by the total number of columns. We consider the annotation of a column correct if a correct label (property) is in the top k labels. We can see that for k=1, it has the probability of 0.0004 (0.04%) to get

[6] two files related to the class person is missing from the classification.

the correct label while our application gets it correctly for 0.34 of the input columns. For k=10, our application gets a correct label 0.91 of the times while getting the correct property randomly is around 0.0004. We discuss the results in Section 4.3.

Table 3: Description of the classification

# numerical columns	319
# understood columns	232
# found in the knowledge graph	137
# classified columns	124

Table 4: Classification score of web data commons tables

Top K-Candidates	Score (FCM)	Random
1	0.34	0.0004
3	0.55	0.0012
5	0.83	0.002
10	0.91	0.004

4.3 Discussion

It is obvious that the scoring results of Olympic Games are much higher than the scores of T2Dv2. Looking closely at the data, we can see that numerical data contained in the Olympic Games CSV files tend to be close to normal distribution (e.g. weight of soccer players). Normal distributions are commonly represented by the mean and variance. Looking at the features we used, we use the mean and the standard deviation, which is the square root of the variance. It is intuitive to see that in distributions that are normal (or close to normal) data are concentrated in the middle (close to the mean). For example, looking at the population density of countries, data are concentrated more to the left, looking close to a chi-squared distribution with k=2. Such numerical properties are harder to classify. It also depends on the distance to other clusters. Some clusters are very close to each other, what results in the points being classified to a nearby but wrong cluster. An example of such cases are the two clusters "areaOfCatchment" and "elevation" of the class dbo:Lake. Sometimes this occurs with unrelevant properties (e.g. wikiPageID or imageSize).
Another thing we noticed is how properties that represent years are akin to understand. Looking at the distribution of values for years, it is hard to guess whether they represent dates of birth of people or dates of football matches. Even with such properties that are easy to mix up, there may be some kind of influence. For example, the birth dates of people are before their death dates. Nonetheless, machine learning techniques often cannot distinguish between birth dates and death dates [18]. The nature of the data used in the model also influences the classification score. For example, if we have two date-related properties, such as birth date (year) of Nobel prize holders, and birth dates (year) of young Internet millionaires, most probably, the classification won't be mixing the two.

The first ones are generally older (less) than the birth dates of the young internet millionaires.

It is also important to note that the results we obtained are not comparable to the results we found in the literature without re-examining their data (when publicly available) and test it on all. The main reason is that most of the results reported in the state-of-the-art (except for [11] and [21]) showed the classification scores for a combination of numeric and non-numeric data sources and did not show the scores of each of them separately. The work by Neumaier et al. [11] is focused on numerical data sources, but the reported results are the scores of the classification of DBpedia's numeric *types* (they divided DBpedia's numeric properties into training and testing sets) and did not perform the test on CSV files. On the other hand, Zhang et al. [21] perform the test on CSV files. They learn from tables rather than from a SPARQL endpoint, and they use context information, such as surrounding text, for matching values and columns. Their approach has a different kind of information to learn from when compared to our approach, which learns from a SPARQL endpoint. To have an accurate comparison between two approaches, both should train on the same training set and test on the same CSV files (test set). This was only feasible in the case of Neumaier et al. [11], which have their software publicly available. We were able to test their approach with our data sets, using the English DBpedia for training. The approach MLSL yields a low classification score, which also concurs with [11] that it is not suitable for classifying CSV files. For Zhang et al. [21] we could not find the software to test it with our data set.

In addition, our approach is domain agnostic in a sense that the endpoint that is queried and the data that is extracted from to create the model is an input, and it does not have to be DBpedia or any specific endpoint. It only has to include data that are of the same *type* that exists in the input CSV files to be classified. Since our approach accepts concepts that are related to the input CSV files, there is no need to explore and include other unrelated concepts in the model. So, only the concepts that matter are included in the model that is used in the classification of the CSV files as shown in the data extraction section 3.1 and the data extraction function Listing 4. Another advantage of our approach is that it does not require numbers to match between the values in the columns and the knowledge graph to label the column with the semantic type. As we focus on how the values reside in the space rather than to which entity these values belong. Moreover, the learning process is semi-automatic (or automatic if the classes are provided, like in the case of Web Tables from Web Data Commons). Despite the fact that concept discovery is not automatic and the user has to provide the concepts as an input to the model creation, the lookup for numerical properties for each concept and the extraction of data for each property is done automatically and included in the model; ergo there is no need to train the model manually by matching columns in CSV files to properties in endpoints.

5 Future Work

There are multiple ways that our work can be extended. One way is to generate R2RML mappings from the annotations that have been obtained. This can be used to generate RDF triples on the fly from the original source files using an application like morph-rdb [13] or ontop [5] while keeping the input files in their original format. Another interesting piece of future work would be to merge duplicate properties due to language differences (e.g. <`http://es.dbpedia.org/property/peso`[7]> and <`http://dbpedia.org/property/weight`>), different subdomains (e.g. <`http://dbpedia.org/property/peso`> and <`http://es.dbpedia.org/property/peso`>) or different naming (e.g. <`http://dbpedia.org/ontology/elevation`> and <`http://dbpedia.org/property/elevationM`>). Our approach can be also extended to suggest the merge of properties that are similar so that two (or more) properties that represent the same thing can be combined in a single property with objects of both. This can be used to improve the classification and to clean endpoints (e.g. generate code that can be applied to the endpoint if such permission is granted). Furthermore, it can be used in the analysis of endpoints and their evolution (e.g. the change in property names), since our approach can detect similar numerical properties. It would be interesting to see if a property is divided or merged (e.g. max temperature per month vs. per year for a given place) and the effects of such actions. Considering the exact match of numbers as in the work of Zhang et al. [21] and combine it with our approach sounds as promising future work.

Acknowledgment We would like to thank EIT Digital for their support. This project has been funded by the Spanish Ministry MINECO and FEDER - project TIN2016-78011-C4-4-R.

References

1. Alobaid, A., Corcho, O.: Olympic games 2020 (2018). https://doi.org/10.5281/zenodo.1408563. URL `https://doi.org/10.5281/zenodo.1408563`
2. Alobaid, A., Corcho, O.: TADA-NumCol (2018). https://doi.org/10.5281/zenodo.1410215. URL `https://doi.org/10.5281/zenodo.1410215`
3. Bezdek, J.C., Ehrlich, R., Full, W.: Fcm: The fuzzy c-means clustering algorithm. Computers & Geosciences **10**(2-3), 191–203 (1984)
4. Cafarella, M.J., Halevy, A., Wang, D.Z., Wu, E., Zhang, Y.: Webtables: exploring the power of tables on the web. Proceedings of the VLDB Endowment **1**(1), 538–549 (2008)
5. Calvanese, D., Cogrel, B., Kalayci, E.G., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Obda with the ontop framework. In: SEBD, pp. 296–303. Citeseer (2015)

[7] which means weight in Spanish.

6. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. https://www.w3.org/TR/r2rml/ (2012). [Online; accessed 26-Oct-2017]
7. Dimou, A., Vander Sande, M.: Rdf mapping language (rml). http://rml.io/spec.html (2014). [Online; accessed 6-Sept-2018]
8. Goel, A., Knoblock, C.A., Lerman, K.: Exploiting structure within data for accurate labeling using conditional random fields. In: Proceedings on the International Conference on Artificial Intelligence (ICAI), p. 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp) (2012)
9. Limaye, G., Sarawagi, S., Chakrabarti, S.: Annotating and searching web tables using entities, types and relationships. Proceedings of the VLDB Endowment **3**(1-2), 1338–1347 (2010)
10. Mihindukulasooriya, N., Poveda-Villalón, M., García-Castro, R., Gómez-Pérez, A.: Loupe-an online tool for inspecting datasets in the linked data cloud. In: International Semantic Web Conference (Posters & Demos) (2015)
11. Neumaier, S., Umbrich, J., Parreira, J.X., Polleres, A.: Multi-level semantic labelling of numerical values. In: International Semantic Web Conference, pp. 428–445. Springer (2016)
12. Pham, M., Alse, S., Knoblock, C.A., Szekely, P.: Semantic labeling: a domain-independent approach. In: International Semantic Web Conference, pp. 446–462. Springer (2016)
13. Priyatna, F., Alonso-Calvo, R., Paraiso-Medina, S., Padron-Sanchez, G., Corcho, O.: R2rml-based access and querying to relational clinical data with morph-rdb. In: SWAT4LS, pp. 142–151 (2015)
14. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. https://www.w3.org/TR/rdf-sparql-query/ (2008). [Online; accessed 11-May-2017]
15. Ritze, D., Lehmberg, O., Bizer, C.: Matching html tables to dbpedia. In: Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics, p. 10. ACM (2015)
16. Ritze, D., Lehmberg, O., Bizer, C.: T2Dv2 Gold Standard for Matching Web Tables to DBpedia. http://webdatacommons.org/webtables/goldstandardV2.html (2015). [Online; accessed 26-June-2018]
17. Syed, Z., Finin, T., Mulwad, V., Joshi, A.: Exploiting a web of semantic data for interpreting tables. In: Proceedings of the Second Web Science Conference, vol. 5 (2010)
18. Taheriyan, M., Knoblock, C.A., Szekely, P., Ambite, J.L.: Learning the semantics of structured data sources. Web Semantics: Science, Services and Agents on the World Wide Web **37**, 152–169 (2016)
19. Venetis, P., Halevy, A., Madhavan, J., Paşca, M., Shen, W., Wu, F., Miao, G., Wu, C.: Recovering semantics of tables on the web. Proceedings of the VLDB Endowment **4**(9), 528–538 (2011)
20. Weise, M., Lohmann, S., Haag, F.: Extraction and visualization of tbox information from sparql endpoints. In: Knowledge Engineering and Knowledge Management: 20th International Conference, EKAW 2016, Bologna, Italy, November 19-23, 2016, Proceedings 20, pp. 713–728. Springer (2016)
21. Zhang, M., Chakrabarti, K.: Infogather+: Semantic matching and annotation of numeric and time-varying attributes in web tables. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp. 145–156. ACM (2013)