Detecting Errors in Numerical Linked Data using Cross-Checked Outlier Detection

Daniel Fleischhacker, Heiko Paulheim, Volha Bryl,
Johanna Völker*, and Christian Bizer

Research Group Data and Web Science, University of Mannheim, Germany
{daniel,heiko,volha,johanna,chris}@informatik.uni-mannheim.de

Abstract. Outlier detection used for identifying wrong values in data is typically applied to single datasets to search them for values of unexpected behavior. In this work, we instead propose an approach which combines the outcomes of two independent outlier detection runs to get a more reliable result and to also prevent problems arising from natural outliers which are exceptional values in the dataset but nevertheless correct. Linked Data is especially suited for the application of such an idea, since it provides large amounts of data enriched with hierarchical information and also contains explicit links between instances. In a first step, we apply outlier detection methods to the property values extracted from a single repository, using a novel approach for splitting the data into relevant subsets. For the second step, we exploit `owl:sameAs` links for the instances to get additional property values and perform a second outlier detection on these values. Doing so allows us to confirm or reject the assessment of a wrong value. Experiments on the DBpedia and NELL datasets demonstrate the feasibility of our approach.

Keywords: Linked Data, Data Debugging, Data Quality, Outlier Detection

1 Introduction

The Linked Data Cloud is constantly growing, providing more and more information as structured data in the RDF format and interlinked between different repositories. Instead of being created and maintained manually, most data sources have their roots in unstructured or semi-structured information available throughout the Web. For example, data sources like DBpedia contain some data extracted from Wikipedia articles. However, though being a major reason for the large amount of Linked Data available, extracting data from unstructured or semi-structured information is error-prone. Even when extracting from semi-structured sources, representational variety (e.g., different thousands delimiters), can lead to problems in the parsing process and finally result in wrong Linked

* Johanna Völker is supported by a Margarete-von-Wrangell scholarship of the European Social Fund (ESF) and the Ministry of Science, Research and the Arts Baden-Württemberg.

Data values. It is unrealistic to manually find errors due to the large amount of data in the repositories, thus automatic means for detecting errors are desirable.

In this paper, we introduce a method for detecting wrong numerical values in Linked Data. First, we determine outliers regarding a single data repository, e.g., on all values assigned by means of the `population` property. For this purpose, we present a way of discovering data subpopulations induced by classes and properties and apply outlier detection to these subpopulations. For example, on the full dataset, the populations of continents would be outliers for the `population` property values since their population values are larger than the predominant population values of cities or countries by several orders of magnitude.

Afterwards, as a second step, we exploit the `owl:sameAs` links of the instances (also called entities) for collecting values for the same property from other repositories. This is an especially important facet of our approach, since it actually uses the links that are a core concept of *Linked* Data which are rarely used in other works (see Sect. 3). If an outlier detected in the first step is only a natural outlier, it does not show up as an outlier in the second step which allows for mitigating the problem of falsely marking natural outliers as wrong values.

In the following, we describe this two-step approach in more detail. We first introduce the foundations of outlier detection in Sect. 2. Afterwards, we give an overview about other works on Linked Data error detection and Linked Data quality in general (Sect. 3). Then, in Sect. 4, we introduce our method for detecting erroneous numerical values in a Linked Data repository paying special attention to the choice of subpopulations of values and cross-checking by means of a second set of data. Afterwards, we evaluate the approach by an experiment on DBpedia and provide the first explorations on the NELL dataset in Sect. 5.

2 Preliminaries

Our approach presented in this paper is relying on the concept of *outlier detection* (sometimes also called *anomaly detection*). In this section, we give an overview of the most important notions used in our work. A more complete overview is given by Chandola et al. [5], where the outlier detection is defined as "finding patterns in data that do not conform to the expected normal behavior".

There can be different reasons for such deviations from the expected behavior. On the one hand, outliers can be caused by erroneous data where the error in the data leads to the actual deviation. On the other hand, there might also exist correct instances which deviate from those patterns, as in the example given above of the population of continents being outliers in the set of population values for cities, countries and continents. Such outliers are sometimes called *natural outliers*. Thus, when using outlier detection for finding errors in the data, special attention has to be paid on how to tell apart such natural outliers from outliers caused by actual data errors.

In all cases, the first step is the discovery of outliers. For this purpose, there are different categories of methods: supervised, semi-supervised, and unsupervised. Supervised and semi-supervised approaches require training data in which

non-outlier values are labeled, and for the supervised approaches outlier values are also labeled. In contrast, unsupervised approaches are independent from such data. Since our approach should be able to work with many different data distributions (e.g., values for population, height, elevation etc.), the creation of training data would be rather expensive, so we only consider unsupervised outlier detection methods. In addition, the methods also differ in their output. Some methods return a binary decision whether a given value is an outlier while other methods provide an outlier score quantifying the degree of ,,outlierness". We only consider the latter group of approaches since they have the advantage that arbitrary thresholds can be chosen to address the trade-off between removing as many actual errors (true positives) vs. removing correct data points (false positives). There are also approaches which consider multiple dimensions of data at once (*multi-variate*) instead of just a single dimension (*univariate*) to improve the detection of outliers by considering values influencing each other. In this work, we only consider univariate approaches because multi-variate methods are more computationally expensive and require a way of determining which value combinations to consider.

In the literature, many different approaches are proposed for unsupervised outlier detection. Some methods assume there is an underlying distribution that generates the data. Values which are improbable according to this distribution, are qualified as outliers. One such approach is to assume an underlying Gaussian distribution of the values, compute mean μ and standard deviation σ values and then mark all values as outliers that are not contained in the interval $[\mu - c\sigma, \mu + c\sigma]$ for a given c. For example, this assumption is backed by the Gaussian distribution's property that 99.7% of all values are within this interval for $c = 3$.

An alternative method for unsupervised outlier detection is the so-called *Local Outlier Factor* (LOF) proposed by Breunig et al. [2]. Compared to other globally working outlier detection approaches, LOF is trying to detect local outliers, i.e., values which deviate from their local neighbors. The idea is that real-world datasets contain data which might not be recognized as a global outlier but its deviation is only recognizable when considering its neighborhood. For this purpose, the LOF algorithm takes a parameter k which defines the number of neighbors to look at. It then determines this number of neighbors and computes an outlier score based on the comparison of the distance of the neighbors to their nearest neighbors with the distance of the currently processed value.

3 Related Work

A number of automatic and semi-automatic approaches for correcting linked data have been proposed, which are either *internal*, i.e., they use only the data contained in the datasets at hand, or *external*, using either additional information sources (such as text corpora) or expert knowledge.

Recent *internal* approaches are mostly concerned with validating object-valued statements (in contrast, our approach targets at numeric literals). The approaches discussed in [9] and [16] first enrich the data source's schema by

heuristically learned additional domain and range restrictions, as well as disjointness axioms, and then use the enhanced ontology for error detection by reasoning. Heuristic approaches for finding wrong dataset interlinks exist, which, for example, rely on finding inconsistent chains of `owl:sameAs` statements [13], or use outlier detection methods [14].

External approaches involve crowdsourcing [1], using platforms like Amazon Mechanical Turk which pay users for micro-tasks, such as the validation of a statement. Another possibility is using games with a purpose to spot inconsistencies as Waitelonis et al. [17] do. *DeFacto* [10] uses a pre-built pattern library of lexical forms for properties in DBpedia. Using those lexical patterns, DeFacto runs search engine requests for natural language representations of DBpedia statements. While it is designed to work on object properties, the approach is transferable to the problem of identifying errors in numerical data as well.

In this paper, we focus on *outlier detection* methods as a means to identify wrong numerical values. This approach is similar to our preliminary approach discussed in [18], but extends it in two respects. First, we identify meaningful subpopulations in a preprocessing step, which makes the outlier detection work more accurately. Second, most of the approaches discussed above do not use *dataset interlinks* at all, despite claiming to be data cleansing approaches for *linked* data. In contrast, we show in this paper that the explicit use of dataset interlinks improves the results of outlier detection, especially with respect to natural outliers.

4 Method

In the following, we describe our overall approach of detecting wrong values in a Linked Data dataset. First, we shortly describe how we determine the properties to check for wrong numerical values before we present the actual process of outlier detection. As discussed above, applying outlier detection to the full dataset might not result in good results since instances referring to different types of real world objects might be contained in the dataset. Thus, we also introduce our way of determining subsets of data to apply the outlier detection on. Finally, we describe the actual detection of erroneous values from the outlier detection results.

4.1 Dataset Inspection

Since we assume no prior knowledge about the dataset, we first have to gather some additional information about it. This step as well as the following steps are most easy to perform when the data is provided by a SPARQL endpoint.

First, we determine the number of instances contained in the repository as well as the names of all properties used in the data. Since we cannot assume to have an OWL vocabulary and its division between object and data type properties available in the dataset, we then determine how often each property is used with a numerical value[1] at the object position. Furthermore, we also

[1] Numerical values are `xsd:int` and `xsd:float` as well as their subtypes.

determine how many distinct numerical values are used with each property by means of the SPARQL query:

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

We then filter the properties to apply outlier detection and remove properties which were only used with a single distinct numerical value. All in all, this process results in a set of properties qualifying for the application of outlier detection.

4.2 Generation of Possible Constraints

Each property is now processed separately in several steps. It is important to note that the wrong value detection is always done for an instance-value pair and not only for an instance since an instance might have several values assigned by means of the same property, e.g., a city having different ZIP codes.

The first step here is to determine the set of constraints that are used to generate subpopulations from the full instance-value set on which a more fine-grained outlier detection is possible which in turn improves the detection of errors. The main motivation behind these constraints is that when always considering the full set of instances, some erroneous values could be *masked* by other values in the dataset while correct values could be erroneously highlighted as being wrong. Masking could for example occur when an instance of the type `Country` has an erroneous `population` count of 400. When considering the whole dataset, this population count would not arouse any suspicion since there are many instances of `Village` with similar population counts. However, when only considering instances of type `Country`, a population count of 400 would be suspicious because hardly any country has such a low population count. Erroneous highlighting of values could occur in the already provided case where instances of the class `Continent` having an actually correct population count are outliers in the dataset of all instances due to the low number of continents and their population counts being much higher than those of countries.

Thus, an important task is to define a way of generating subsets of the full instance set. In this work, we do this generation by applying constraints to the set of instances so that only those instances are retained which fulfill the constraints. We propose three different types of constraints:

- *Class constraints:* A class constraint on class C applied to an instance set limits it to instances which belong to this class.
- *Property constraints:* A property constraint p limits the instances to those connected to an arbitrary object (instance or data value) by means of p.
- *Property value constraints:* A property value constraint is defined by a property p and a value v which can be either an instance or a data value. It limits the instances to those which are connected to a value v by means of p.

Class constraints as also applied by [18] are the most obvious way of utilizing the class structure already contained in the dataset. They allow capturing the masking for the `population` property described before.

In cases where the class structure is not detailed enough, the two additional constraint types can help to compensate these shortcomings. In real world datasets, property constraints can help to deduce statements about an instance's missing type [15]. For example, given a class `Vehicle` and a property `maximumAltitude`, this property can compensate for a missing class assertion to `Aircraft` and thus allow to detect, e.g., too high `weight` values for the instances that could otherwise be masked by other `Vehicle` instances such as ships. The choice of which properties to use as constraints is based on the number of usages in the current instance set. When even the property constraints are not able to provide a sufficiently fine-grained division into subpopulations, property-value constraints can be used. An example for such a constraint is the property `locatedIn` with the value `UnitedArabEmirates` (UAE). Since the average temperature in the UAE is higher than the temperature in most other countries, a too low `averageTemperature` assigned to a city in the UAE could be masked by cities from other countries. When only considering cities from the UAE, the low average temperature is suspicious and thus detectable as being erroneous.

Both property-based constraints share the problem that they might introduce a high number of constraints since the number of properties might be much higher than the number of classes used in the dataset. This can lead to higher computational effort for choosing the constraints. This effort is even higher for property-value constraints that do not only require to examine the used properties but also the values connected to instances by means of these properties.

4.3 Finding Subpopulations

Applying outlier detection to all of the potentially many subpopulations which can be defined on a dataset is impractical especially because the runtime of outlier detection algorithms heavily depends on the number of values they are applied on. Hence, we introduce an intermediate step for determining the most promising subpopulations to apply outlier detection on.

The exploration is organized in a lattice as shown in Fig. 1 similar to the one used by Melo et al. [12]. Each node of the lattice is assigned a set of constraints which determines the instances considered at this node. The root node has the empty constraint set assigned and thus represents all instances and corresponding values of the currently considered property. For this set of instances, we compute a histogram which represents the distribution of values in the subpopulation. Starting with the root node, our approach manages a queue of all not yet extended nodes and thus extends the lattice in a breadth-first-manner.

When processing a node from this queue, we create its child nodes, each having an additional constraint compared to the parent node. The additional constraints are those from the set of possible constraints which are not yet used in the parent node. If a node for the resulting set of constraints already exists in the lattice, we do not consider the new node further. Otherwise, we determine the instances which adhere to this new set of constraints and compute the histogram of the value distribution. Based on this value distribution, we enforce a set of pruning criteria to keep the search space clean which helps us to determine

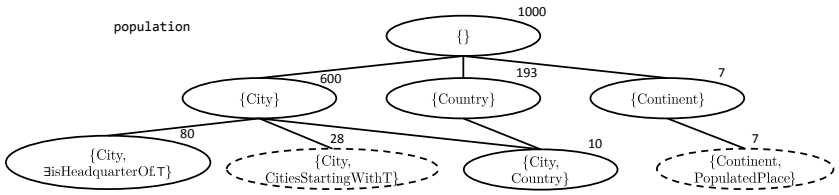


Fig. 1: Example for subpopulation lattice for property `population`. Numbers to the upper right of a node give the number of instances fulfilling the constraint set. Dashed nodes would be pruned, the left one for too low KL divergence, the right one for not reducing the instance set further.

interesting subpopulations independently from any further knowledge about the constraints and their relation to each other. In particular, we prune subpopulations which only contain a low number of instances or maybe no instances at all since those are considered to be too specific.[2] As another criterion, we consider the instance reduction ratio, i.e., the change ratio in the number of instances of the new node compared to its parent node. If the additional constraint leads to a reduction of less than 1%, our approach prunes the node. For instance, this case could occur when adding a class constraint on `PopulatedPlace` to a constraint set which was previously also constrained on `Continent`.

Finally, we compute the Kullback-Leibler (KL) divergence [8] between the discrete value distributions represented by the histograms of the new node and the parent node. If the divergence is lower than a given threshold, we assume the additional constraint to be independent from the previously applied constraints, i.e., the actual distribution of values was not changed but only the number of instances. In these cases, an outlier detection run on the newly created set of instances would not yield additional insights and thus we prune those nodes. For example, this pruning could happen when adding a class constraint on the class `NamesStartingWithT` to a constraint set for a property representing the population count. Since each additional constraint leads to a smaller number of instances compared to the parent node, the sampling error might also influence the resulting KL divergence value. To address this effect in our considerations, we normalize the values using the number of instances of the resulting node leading to the formula

$$\text{divergence}(\mathit{parent}, \mathit{child}) = \left| \frac{|\mathit{child}|}{|\mathit{parent}|} \cdot \sum_{i=1}^{B} \ln\left(\frac{h_{\text{parent}}(i)}{h_{\text{child}}(i)}\right) h_{\text{parent}}(i) \right| \tag{1}$$

where *parent* and *child* are the nodes of the lattice, $|n|$ the number of instances for a node n and h_{parent} as well as h_{child} the histograms representing the respective value distribution which each have B bins. Furthermore, we also apply Laplace smoothing to the histograms. We assume a higher divergence to show a more important change in the distribution of values and thus being more inter-

[2] In our experiments, a value of 5 was used.

esting for the further processing. Based on this assumption, we prioritize nodes having a higher KL divergence to their parents in later expansion steps, as well as in cases where too many nodes would have to be expanded, we limit the expansion to the highest ranked nodes.

4.4 Outlier Detection and Outlier Scores

After the lattice has been determined, we perform outlier detection on all unpruned nodes of the lattice and store the resulting outlier scores together with the set of constraints which led to the corresponding instance set.

As soon as the outlier detection run is completed on the property, we have a list of instance-value combinations with a set of pairs, consisting of a constraint set and an outlier score. One advantage of having multiple outlier scores compared only a single outlier score for each instance-value combination is the possibility to apply different weighting schemas to the scores to combine them into a single assessment for each instance-value pair. At this point, it is also possible to further consider an ontology schema possibly contained in the dataset. For example, outlier scores for class constraints of more specific classes can be assumed to have more significance than those for constraints to more abstract classes and can thus be weighted higher. In particular, we explore a measure which assigns an instance-value combination with the outlier score of the constraint set containing the most specific constraint according to the hierarchy which performed best in our pre-studies in combination with the LOF outlier detection approach. It is noteworthy, that too specific constraint sets are already filtered during the creation of the subpopulation lattice which prevents us from choosing the outlier scores generated for such subpopulations. For determining the specificity of an entity in the hierarchy, we use property paths as introduced in SPARQL 1.1 like in the following query for a class specified by its IRI `CLS`

```
SELECT COUNT(DISTINCT ?i) AS ?cnt WHERE {<CLS> rdfs:subClassOf+ ?i}
```

This query provides us with the number of direct and indirect super classes of the given class which serves as an estimate for its specificness.

4.5 Cross-checking for Natural Outliers

As described in Section 2, values may not only be detected as outliers when they are wrong but also if they are natural outliers in the considered dataset. To prevent this false detection, we apply an additional cross-checking step to the results of the first outlier detection.

One of the unique selling points of Linked Data is the interlinking of datasets. Using URIs to point to resources in remote repositories, it is possible to specify for an instance which equivalent instances can be found in other repositories. Given that in the Linked Data and Semantic Web community the reuse and interlinking of schema vocabularies is encouraged, these equivalence assertions allow us to retrieve additional property values for the same instance. Even if the vocabulary is not reused, ontology matching techniques [7] can enable the retrieval of additional property values by determining equivalent properties to the

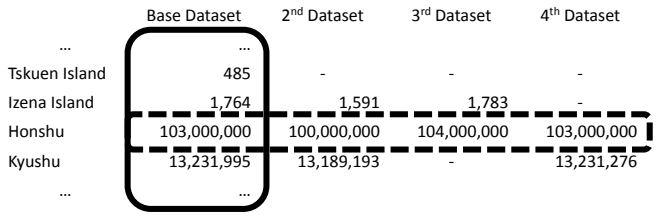


Fig. 2: Using two independent outlier approaches for the DBpedia property `populationTotal` and the instance "Honshu" to improve the detection result. Only considering the base dataset (vertical), the actually correct value is detected as an outlier. The detection run on the values from different sources (horizontal) confirms the value and thus prevents to mark the value as a wrong value.

currently relevant property. For the special case of DBpedia and its versions in several languages, inspections [3] revealed that the number of instances described in multiple datasets is relatively low. But even if the additional data is sparse, we assume that natural outliers are often more interesting for humans and hence more often described in several datasets (e.g., the highest mountain is probably described in more datasets than some arbitrary "non-special" mountain).

Using this feature of Linked Data, we have a way of compensating problems introduced by natural outliers. By gathering additional property values for an instance it is possible to test the value found in the current dataset for its "outlierness" in this second set of data. Since these values are expected to be the same if all values are fully correct, it is sufficient to assume a normal distribution for the values and check whether a value lies within a given number of standard deviations around the mean value (cf. Sect. 2). If the assessed value lies within the interval around the mean, the probability is high that the value is only a natural outlier and thus is not an erroneous value. We only consider values as wrong if they are outliers in both detections. This principle is depicted in the real-world example in Fig. 2 where an outlier detection based on the vertical axis would lead to a detection as a wrong value while the second outlier detection run on the horizontal axis confirms the population value in the base dataset.

5 Experiments

For testing the approach described in the previous section, we performed an evaluation on DBpedia[3] and its language versions which we present in detail in the following. Furthermore, we report on an evaluation on the NELL dataset in combination with cross-checking on several Linked Data sources.

5.1 DBpedia Experiment

The first experiment was performed on the DBpedia 3.9 dataset. DBpedia [11] is a large scale structured multi-lingual cross-domain knowledge base automat-

[3] `http://dbpedia.org`

ically extracted from Wikipedia. The current version of DBpedia contains 2.46 billion facts describing 12.6 million unique things, and is a widely used high-impact knowledge resource with around 5,000 downloads a year. The data is extracted from Wikipedia infoboxes (tables usually found in upper right part of a Wikipedia page), page categories, interlanguage links and many more, which are automatically parsed to extract facts like "population of Mannheim is 314,931".Data is extracted from 119 Wikipedia language editions, and is represented as a distinct language edition of the knowledge base.

We let the approach run on the whole dataset, generating ranked lists of possibly wrong values for each property. As an outlier detection algorithm, we used the Local Outlier Factor in the implementation provided by the Rapidminer[4] Extension for Anomaly Detection.[5] The k parameter of LOF was set to 10 resp. to the number of values if there were less than ten. Experiments using different number of bins for the histogram generation turned out that the single KL divergences between children and parent nodes had more variance for higher number of bins. This increased variance led to a more exact detection of similar distributions and thus more pruning in the lattice. However, increasing the number of bins further also increased the runtime of the lattice generation without leading to an adequate reduction of the outlier detection runtime and without clear improvements in the error detection. Thus, we used 100 bins as a compromise between exactness of pruning and runtime. The generation of subpopulations was done on the YAGO[6] classes assigned to the instances. The YAGO classes are very fine-grained (e.g., there is a class `CitiesAndTownsInAbruzzo`) which allows us to only work with class constraints in this experiment.

For the cross-checking of outliers by means of additional instance data, we used the multi-lingual data contained in the DBpedia dataset. This data is the result of different Wikipedia language versions describing the same things which leads to multiple DBpedia instances representing these things throughout the DBpedia language versions. Notably, the entity overlap across languages is not high: out of 2.7 million instances of the 17 most populated DBpedia ontology classes,[7] 60% are described (i.e., have at least one property) only in one language (predominantly English), and only around 23% of all entities are described in three or more languages. Note that we consider only those language editions for which infobox types and attributes are mapped to classes and properties of the DBpedia ontology. In DBpedia 3.9, mappings which were manually created by the DBpedia community for 24 languages were used for data extraction. In the datasets based on these mappings the same property URIs are used across languages: e.g., the DBpedia ontology property `populationTotal` is used for the population property of a populated place in, e.g., German or French editions even if the original Wikipedia infoboxes use language-specific attribute names.

[4] `http://rapidminer.com`
[5] `https://code.google.com/p/rapidminer-anomalydetection/`
[6] `http://www.mpi-inf.mpg.de/yago`
[7] `http://wiki.dbpedia.org/Datasets39/CrossLanguageOverlapStatistics`

Table 1: Inter annotator agreement observed for property samples and number of correct instance-value combinations according to majority of annotators

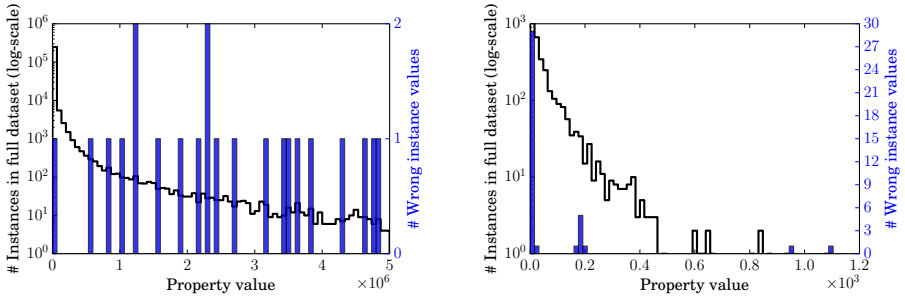
	`elevation`	`height`	`populationTotal`
Observed agreement	0.987	0.960	0.960
Fleiss' κ	0.968	0.916	0.917
# correct	69	60	57

To assess the performance of our approach for detecting wrong values, we chose the three DBpedia ontology properties: `height`, `elevation` and `populationTotal`. From each of the three ranked lists, we randomly sampled 100 instance-value combinations where we introduced a bias towards possibly wrong combinations by scaling the selection probability proportionally to the score determined by the outlier detection. The resulting values have been independently reviewed by three human annotators regarding the correctness of the values. For determining the correctness of a value, a typical process of the annotators was to first have a look at the current Wikipedia page describing the instance. Additionally, the Wikipedia page in its version as of the time of the extraction run was inspected. Using these two sources, it was possible in most cases to recognize errors in the values which stemmed from parsing errors or vandalism. If these inspections did not yet lead to the detection of an error, the most promising non-English Wikipedia articles about the instance were consulted, e.g., the article in the language most related to the instance. Finally, cited external sources were consulted or the annotators tried to find reliable information on the Web using search engines. If no proof for an error in the data was found, the instance-value combination was marked as correct, otherwise as wrong.

We computed the inter annotator agreement (IAA) between the three annotators on the evaluated lists by means of Fleiss' kappa.[8] The results of the IAA analysis are shown in Table 1. These values show a very high agreement for all three properties. From a short analysis of the few disagreements, we discovered that most of these were caused by an annotator not finding the relevant external information to assess the correctness of the value. Also the table shows the number of correct values in the datasets used for evaluation. It is important to note that, due to the way we sampled the example instances, these values are not able to provide an unbiased insight into the correctness of DBpedia but are overstating its incorrectness.

Furthermore, we plotted the distribution of the wrong instance-value combinations discovered during the manual evaluation and the actual value distribution not only over the sampled values but over all values in the dataset. These diagrams provide us with important knowledge about the erroneous values. For example, in Fig. 3b we see that there are two spikes of erroneous values. The first is located at the lower bound of the value range and mostly contains errors for

[8] We used the tool at `https://mlnl.net/jg/software/ira/` for computing IAA.

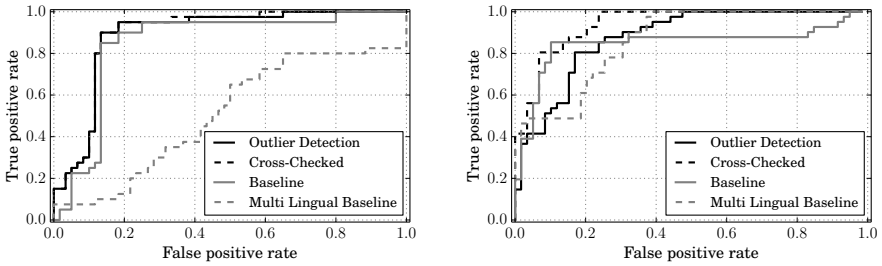


(a) populationTotal (b) height

Fig. 3: Distribution of all values in dataset (in log-scale) and erroneous values discovered in the manual evaluation for the different properties. Property value and all instance count scale restricted to the given ranges.

entities of the class `Person` caused by using the wrong unit (1.98 cm instead of 1.98 m) and also values which are wrong but not directly recognizable as errors because they fit the usual height of people. The second spike is located around a value of 200 and again results from using the wrong unit in this case 198 m instead of 1.98 cm. This finding especially confirms the need for using subpopulations of the data instead of the full dataset since we see from the overall data that values close to 200 not directly point to data errors (e.g., for buildings this value is totally possible). The two other properties both show the erroneous values to be distributed relatively homogeneously as illustrated by Fig. 3a and not only found to be corner cases in the given ranges. These errors would not be recognizable without considering subpopulations of the data.

Based on these manually annotated value lists, we determined the performance of our approach with and without cross-checking as described in Section 4.5. For each evaluated property, we also provide two baseline values. The first baseline, which we identify by "Baseline", is computed by determining the median of all values and then computing the absolute difference between this median and the current instance's value. We use the resulting value as a score for the value being wrong. The second baseline (referred to as "Multi-lingual baseline") uses the multi-lingual data also employed by the cross-checking. For getting a score for an instance-value combination, we retrieve all values available for languages other than English. For two or more values, we compute the score for a value v as $|v - \mu|/\sigma$ where μ is the mean of the non-English values and σ their standard deviation. Assuming a normal distribution of the values, this means that approx. 95% of the values should have a score less or equal to 2. If we only retrieve zero or one value, we assign a score of 2. This fall-back value has been chosen since values for which not enough information is available in the multi-lingual dataset are more probable to be erroneous than values for which



(a) height (b) populationTotal

Fig. 4: ROC curves for evaluated error detection methods

we find values which validate their correctness. In the cross-checking step, we consider all values with a score of at least 2 as outliers.

We plotted the receiver operating characteristic (ROC) curves for each property using the computed scores to rank the instance-value combinations. The results for the properties `height` and `populationTotal` are shown in Fig. 4a and 4b, each containing the results for the raw detection approach, the filtered one and the two baselines. We also computed the area under the curve (AUC) for each property and each approach. The results are provided in Table 2.

Table 2: Area under the curve determined for the given samples and approaches

Approach	`elevation`	`height`	`populationTotal`
Outlier Detection	**0.872**	0.888	0.876
Cross-Checked Outlier Detection	0.861	**0.891**	**0.941**
Baseline	0.745	0.847	0.847
Multi-lingual Baseline	0.669	0.509	0.860

First of all, we see that the AUC values of the cross-checked outlier detection approach are better than the baselines for all three properties. This approach is also superior to our non-cross-checked approach for `height` and `populationTotal`. Only for `elevation` it is slightly worse. Closer evaluation of this decrease showed that it was caused by a wrong value for elevation contained not only in the English dataset but also in the multi-lingual data. This duplication of wrong values could be caused by people copying values from one Wikipedia page to another one without checking the values. However, during the evaluation, this was not a frequent problem, and if it occurs more often for other datasets, a possible solution would be to employ copy-detection approaches [6].

For the property `height`, the difference between baseline methods and our methods is considerably smaller. This fact seems to be caused by a large number of persons in the example dataset. The median value used by the baseline is the height of one (correct) person instance. Since the `height` property for persons follows a normal distribution as also reported in [18], the median deviation

Table 3: Numbers of values found for different NELL instances

Number of values	1 (only NELL)	2	3	4	5	Total
Number of instances	6,187	5,043	3,144	6,471	13,100	33,946

works especially good and returns low scores for person instances. Although this behavior leads to high scores for the non-person instances, it gives a strong baseline for our dataset. Another interesting detail is that the multi-lingual baseline does not perform too well which is due to 86 instances not having enough multi-lingual data to assess their correctness. The greatest part of these instances is made up by the person class, especially by athletes of sports mostly famous in English-speaking countries like rugby and baseball and seemingly not exhaustively described in other languages. Due to this fact, the cross-checking step hardly improves the already good results of the base approach.

Finally, for the `populationTotal` property, the baseline performs well in the first parts of the examples, where it even outperforms the basic outlier detection approach. However, since the baseline does not perform constantly well on the data, the final AUC value for the outlier detection approach is higher. As we can also derive from the multi-lingual baseline's comparably high AUC, there is more data available in the different language versions than for the other properties. Nevertheless, for 60 values there is not enough information for assessing the correctness. The higher availability of multi-lingual data also leads to a clear increase for the cross-checking method and makes it the clearly best performing approach on this dataset. Furthermore, it demonstrates the advantages of combining two orthogonal detections to reach a final correctness decision.

All in all, we see that the cross-checked method performs consistently well for all three properties. It always produces better results than the baseline approaches. Most of the time it is also better than the non-cross-checked approach showing that it indeed prevents natural outliers from being detected as errors.

5.2 NELL Experiment

For the second experiment, we let our approach run on the NELL dataset [4] in its RDF version [19]. The NELL dataset is produced by crawling the Web and extracting structured data out of the discovered unstructured information. Given this extraction method, we can assume that parsing errors and other difficulties result in some quality deficiencies in the data. We let our approach examine the latitude and longitude values contained in the RDF version of NELL and try to find wrong values in it. For getting data to cross-check the values, we used the Wikipedia links contained in the NELL data to the corresponding DBpedia instance. Besides the DBpedia values for longitude and latitude, we used the `owl:sameAs` links assigned to the DBpedia instances to find further instances in the Linked Data cloud which provided the desired values. We included the values we could retrieve from Freebase, GeoNames, YAGO and DBpedia. Statistics on the number of values we were able to find are shown in Table 3. These numbers

demonstrate that it is possible to gather additional values from the Linked Data cloud to enable the cross-checking of detected outliers and to clean up the data.

However, during the actual run of the outlier detection only few values with a sufficiently high outlier score showed up. An inspection of the data from the other repositories, and for some instance values also an inspection using a web-based map service, showed that there is close to no deviation throughout the datasets. Almost all of the inspected values were correct possibly because of the highly standardized value format for latitude and longitude which leads to only few parsing errors. The small deviations of the values seem to be caused by subjective decisions, e.g., where to exactly position the longitude-latitude marker for the area of a county. Nevertheless, the latitude value with the highest outlier score which was not filtered by the cross-checking showed to be a data error. Being assigned to the NELL instance `http://nell-ld.telecom-st-etienne.fr/county_grey_county`, the latitude value was detected to be wrong also based on its outlierness for the population of the class `County`. An inspection of the Wikipedia page assigned by NELL showed that it should actually represent Grey County, Ontario, Canada[9] whereas the coordinates provided by NELL are in the area of Greymouth, New Zealand which belongs to the Grey District.[10] This hints to disambiguation problems. This result is in line with the findings of Paulheim [14] who also discovered that NELL has problems with homonyms when linking data. In this special case, the confusion could have been amplified by the near synonymy of district and county. All in all, though not finding greater amounts of data errors, we think this use case demonstrates the availability of data from different repositories and thus the applicability of cross-checking for improving wrong value detection.

6 Conclusion

In this work, we presented our approach for detecting wrong numerical values in Linked Data. The main contribution of our work is that we are especially taking advantage of the core concepts of Linked Data: links and vast amounts of data. By following `owl:sameAs` links for instances, we gather additional data for the same facts which we then use to cross-check the assessment of correctness gained during a first outlier detection run on a single repository. This procedure allows us to better handle natural outliers and thus reduce the false positive rate. In addition, we also presented a lattice-based method of detecting interesting subsets of values to apply outlier detection to. The performance of our approach was assessed on DBpedia and we also showed the applicability of cross-checking on more general repositories, here represented by the NELL dataset.

In future work, we will consider additional value types for checking correctness like dates. Furthermore, we will investigate the possibility of efficiently finding pairs of values on which multi-variate outlier detection can be applied. We also plan to gather human feedback on the validity of detected errors and use this

[9] `http://en.wikipedia.org/wiki/Grey_County`
[10] `http://en.wikipedia.org/wiki/Grey_District`

feedback to investigate the possibilities of learning more promising combinations of different weighting schemes.

References

1. Acosta, M., Zaveri, A., Simperl, E., Kontokostas, D., Auer, S., Lehmann, J.: Crowdsourcing Linked Data quality assessment. In: ISWC 2013 (2013)
2. Breunig, M.M., Kriegel, H.P., Ng, R.T., Sander, J.: LOF: Identifying density-based local outliers. SIGMOD Rec. (2000)
3. Bryl, V., Bizer, C.: Learning conflict resolution strategies for cross-language Wikipedia data fusion. In: Proc. of the WebQuality Workshop at WWW'2014 (2014)
4. Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Jr., E.R.H., Mitchell, T.M.: Toward an architecture for never-ending language learning. In: Proc. of the 24th AAAI Conference on Artificial Intelligence (2010)
5. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM Comput. Surv. (2009)
6. Dong, X.L., Berti-Equille, L., Srivastava, D.: Truth discovery and copying detection in a dynamic world. Proc. VLDB Endow. (2009)
7. Euzenat, J., Shvaiko, P.: Ontology Matching, Second Edition. Springer (2013)
8. Kullback, S., Leibler, R.A.: On information and sufficiency. The Annals of Mathematical Statistics 22(1), 79–86 (03 1951)
9. Lehmann, J., Bühmann, L.: ORE – a tool for repairing and enriching knowledge bases. In: ISWC 2010. Springer (2010)
10. Lehmann, J., Gerber, D., Morsey, M., Ngomo, A.C.N.: Defacto-deep fact validation. In: ISWC 2012. Springer (2012)
11. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia - a large-scale, multilingual knowledge base extracted from Wikipedia. Semantic Web Journal (2014)
12. Melo, A., Theobald, M., Völker, J.: Correlation-based refinement of rules with numerical attributes. In: Proc. of the 27th International Florida Artificial Intelligence Research Society Conference (2014)
13. de Melo, G.: Not quite the same: Identity constraints for the web of linked data. In: Proc. of the 27th AAAI Conference on Artificial Intelligence (2013)
14. Paulheim, H.: Identifying wrong links between datasets by multi-dimensional outlier detection. In: 3rd International Workshop on Debugging Ontologies and Ontology Mappings (WoDOOM) (2014)
15. Paulheim, H., Bizer, C.: Type inference on noisy RDF data. In: ISWC 2013 (2013)
16. Töpper, G., Knuth, M., Sack, H.: DBpedia ontology enrichment for inconsistency detection. In: Proc. of the 8th International Conference on Semantic Systems (2012)
17. Waitelonis, J., Ludwig, N., Knuth, M., Sack, H.: Whoknows? evaluating linked data heuristics with a quiz that cleans up DBpedia. Interactive Technology and Smart Education (2011)
18. Wienand, D., Paulheim, H.: Detecting incorrect numerical data in DBpedia. In: ESWC 2014 (2014)
19. Zimmermann, A., Gravier, C., Subercaze, J., Cruzille, Q.: Nell2RDF read the web, and turn it into RDF. In: Proc. of the 2nd International Workshop on Knowledge Discovery and Data Mining Meets Linked Open Data (2013)