Detecting Incorrect Numerical Data in DBpedia

Dominik Wienand[1] and Heiko Paulheim[2]

[1] Technische Universität Darmstadt
Knowledge Engineering Group
dominik.wienand@online.de
[2] University of Mannheim, Germany
Research Group Data and Web Science
heiko@informatik.uni-mannheim.de

Abstract. DBpedia is a central hub of Linked Open Data (LOD). Being based on crowd-sourced contents and heuristic extraction methods, it is not free of errors. In this paper, we study the application of unsupervised numerical outlier detection methods to DBpedia, using Interquantile Range (IQR), Kernel Density Estimation (KDE), and various dispersion estimators, combined with different semantic grouping methods. Our approach reaches 87% precision, and has lead to the identification of 11 systematic errors in the DBpedia extraction framework.

Keywords: #eswc2014Wienand, Linked Open Data, DBpedia, Data Quality, Error Detection, Outlier Detection, Clustering

1 Introduction

DBpedia [10] is a central hub of the Linked Open Data Cloud [2]. Its goal is to make structured data from Wikipedia available to the Semantic Web. In its current version,[3] DBpedia[4] contains information about more than 4.0 million things, including 832,000 persons, 639,000 places, 372,000 creative works, 209,000 organizations, and 226,000 species.[5]

Given its approach of heuristic information extraction from a crowd-sourced web site, DBpedia contains various kinds of errors [18]. Data is entered and maintained manually in Wikipedia, and the input is neither restricted nor validated automatically. This makes it prone to both factual errors and problems during parsing, e.g., if number formats or units of measurement are used which are not expected by the DBpedia extraction code.

While DBpedia deals with various kinds of information, given as classes, instances, and relationships, this paper focuses on the detection of errors in the

[3] DBpedia version 3.9, which has been released on September 17th, 2013
[4] Unless otherwise indicated, all statements about the DBpedia knowledge base refer to version 3.8. Many of the errors reported in this paper have been fixed for the 3.9 release due to the fact that we were able to identify them with methods discussed in this paper.
[5] http://dbpedia.org/About

primitive numerical attributes, using outlier detection and clustering. That is, given one property, such as `dbpedia-owl:populationTotal`,[6] representing the population of a place, we want to detect wrong values that are used as literal objects of that property. We focus on *unsupervised* methods, i.e., methods that do not use domain knowledge such as typical ranges of attributes.

Outlier detection is the method of finding an observation "that appears to deviate markedly from other members of the sample in which it occurs." [4] An outlier may be caused by an error in the data, as well as represent an unusual, but correct value. For example, in a series of country populations in the order of magnitude of millions, a value larger than a billion may be wrong, or refer to unusually large countries, such as China or India. However, in many cases, outliers are caused by wrong data points. Therefore, outlier detection can be used as a means to detect errors in data.

The rest of this paper is structured as follows. Our approach, as well as the methods used for clustering and outlier detection, is sketched in section 2. We show the evaluation of our approach in both a pre-study with a selection of prominent attributes from DBpedia, as well as on a random sample of resources in section 3, and we discuss systematic sources of errors identified in DBpedia in section 4. We wrap up with a review of related approaches in section 5, and an outlook on future work in section 6.

2 Approach

As discussed above, simple outlier detection approaches are limited by the existence of natural outliers. Consider a property such as `dbpedia-owl:populationTotal`, which represents the total population of a `dbpedia-owl:PopulatedPlace`. This includes villages, towns, cities, states, countries, continents and – contrary to the label – also some unpopulated places such as ghost towns and uninhabited islands. That means that most countries and continents will appear to be outliers by most metrics because they are only few in number, but exceed the population of the villages, towns and cities, that make up the majority of the entries, by far.

To cope with that problem, we propose a two-step approach: first, we group the subjects by their types – in the example, separating villages, cities, countries, etc. – and then apply outlier detection to those groups in isolation in order to obtain a more robust error detection.

2.1 Grouping Subjects

Many resources in DBpedia have one or more types, which we can utilize to separate subjects. The most basic way of doing that is to group the subjects of

[6] The following namespace conventions are used in this document: `dbpedia=http://dbpedia.org/resource/`, `dbpedia-owl=http://dbpedia.org/ontology/`, `dbpprop=http://dbpedia.org/property/`, `owl=http://www.w3.org/2002/07/owl`

each property based on each RDF type found in the set of subjects. However, not all types are actually useful for the outlier detection process and some are actually detrimental.

Since in OWL everything is an instance of `owl:Thing`, the subset containing all subjects of type `owl:Thing` will generally contain all subjects of the original set and therefore not provide any further insight. The same can be true for other types (e.g., `dbpedia-owl:PopulatedPlace` for `dbpedia-owl:populationTotal`), so when grouping by single types, it is advisable to first check if the group represents a significantly smaller subset before applying any further outlier detection methods.

Another problem to cope with is the presence of faulty types. Sometimes, types are missing in DBpedia, in other cases, types are wrongly assigned. [12] A typical example are the types `dbpedia-owl:Village` and `dbpedia-owl:City`, which are not uniformly used: there are instances of `dbpedia-owl:Village` with a population over 100,000 inhabitants, as well as instances of `dbpedia-owl:City` with less than 10 inhabitants. Therefore, relying on single types for grouping the subjects of examination can lead to problems.

Since the missing and wrongly assigned types are not equally distributed across all schemas used in DBpedia (e.g., DBpedia, UMBEL, and YAGO), we consider another preprocessing strategy, i.e., clustering by type vectors. For this approach, we consider all types of a subject as a vector of boolean values, representing whether or not the subject is of a certain type, and then apply traditional clustering techniques to subjects stored in this vector representation. To create these vectors, we use the *FeGeLOD* framework [13], which is designed to automatically enrich resources with information gathered from Linked Open Data. FeGeLOD first collects the information for all subjects, creating a binary feature for each type. Then, we apply a threshold p to filter out features that are either too generic, appearing in over $p\%$ of all cases, or too specific, appearing in less than $1 - p\%$ of all cases. The actual clustering is done with the Estimation Maximization (EM) algorithm [3], using the implementation in *WEKA* [5].

2.2 Outlier Detection Approaches

Classical outlier detection approaches assume an underlying distribution (usually a normal distribution). The basic method of those classical approaches is that values that do not fall into the assumed distribution are outliers.

However, those approaches are unsuitable for our purposes, because the data we are dealing with often does not meet those assumptions. Most importantly, the assumption of a normal distribution is not suitable for the vast range of different datasets found in DBpedia. For example, the population sizes of cities follow a log-normal rather than a normal distribution, i.e., there are many more small cities than there are very large cities. Further problems arise with regards to methods being designed for small sample sizes, or only being able to detect one or a few outliers at a time.

More recent outlier detection approaches, which are not that dependent on the assumption of a normal distribution, are based on robust statistics. One sim-

ple such approach is based on the Interquartile Range (IQR). For this method, three points are defined: the median of all values is the 2-quartile (Q_2), the median of those values smaller than Q_2 is Q_1, and the median of those values larger than Q_2 is Q_3. The interquartile range IQR is then defined as the distance between Q_3 and Q_1. Traditionally, every data point smaller than $Q_1 - 1.5 \cdot IQR$ and every point larger than $Q_3 + 1.5 \cdot IQR$ would be considered an outlier, as this roughly corresponds to three standard deviations from the mean for a normal distribution. This approach can be generalized to use the distance between arbitrary subdivisions of the data set, such as $P_{95} - P5$ for percentiles. Also, for our purposes, we will need to use much larger factors than 1.5 due to the heterogeneous nature of the data in DBpedia. The two main parameters of IQR are thus the factor used, and the number of percentiles.

Other approaches are based on estimating the center of the distribution and a range of assumed to be valid values. The median is the most common way of estimating the center of the population. The range of valid values can be estimated in various ways. The classic approach to estimating the dispersion of a population in a robust way is the Median absolute deviation (MAD), defined as $MAD = median_i(|X_i - median_j(X_j)|)$. That is, we first calculate the distance from each point to the median of the dataset and then take the median of those values as the measure of dispersion. As for IQR, a constant factor determining the allowed distance from the median for values considered as non-outliers is the main parameter.

An approach to outlier detection that is not based on robust statistics utilizes Kernel Density Estimation (KDE) [11]. Let $x_1, x_2, ...x_n$ be independent and identically distributed (iid) random variables, drawn from a distribution with an unknown density function f, then

$$f_h(x) := \frac{1}{nh} \sum_{i=1}^{n} K(\frac{x - x_i}{h}) \tag{1}$$

is the kernel density estimator of f, where h is a smoothing factor called *bandwidth*, and K is a so-called Kernel, a symmetric, non-negative function that integrates to 1. For our purposes, we will use the Gaussian normal distribution, $\frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$, using the data sample's mean (μ) and standard deviation (σ), which satisfies all requirements of a kernel. The bandwidth h can be chosen according to "Silverman's rule of thumb" [14] as $(\frac{4\hat{\sigma}^5}{3n})^{\frac{1}{5}}$, where $\hat{\sigma}$ is the sample standard deviation. This bandwidth has been shown to yield optimal results for cases where the underlying distribution is actually normal and reasonable results for unimodal, symmetric distributions. [6]

To calculate outlier scores for a given dataset, we first create a KDE from the data and then calculate the resulting probability at each point. To put this probability into relation we compare it to the mean probability over all points, $mp = \frac{1}{n} \sum_{i=1}^{n} \hat{f}_h(x_i)$. The relative probability of one data point being normal is then $rp(x) = \frac{\hat{f}_h(x)}{mp}$, where $rp(x) > 1$ indicates an above average probability, $rp(x) < 1$ indicates a below average probability. To obtain a binary classification, a threshold is applied, e.g., all x with $rp(x) < 0.1$ are considered as outliers.

As naïve implementations that evaluate the KDE at every input point individually can be inefficient on large datasets, implementations based on Fast Fourier Transformation (FFT) have been proposed. [15] Those implementations allow sampling the function only at equidistant points, but offer a performance that is orders of magnitudes faster than naïve implementations. A million samples can be evaluated in about one second, while evaluating the same number of data points individually would take hours. However, rounding to the nearest available sample usually leads to a loss in precision.

Furthermore, since KDE is not an inherently robust method, outliers can affect the outcome. Thus, an *iterative application* of the approach, i.e., detecting outliers with KDE, removing the outliers, and re-running the process on the remaining data, often improves the results. [7]

3 Evaluation

We perform a two-fold evaluation. First, we conduct a pre-study with three selected attributes. Then, we evaluate the performance of the best performing methods on a random sample of DBpedia.

3.1 Pre-Study

We conduct a pre-study on three properties, `dbpedia-owl:populationTotal`, `dbpedia-owl:height`, and `dbpedia-owl:elevation`, to assess their usefulness with regard to typical data provided by DBpedia. These predicates were selected due to their high coverage, as well as their diverse use (for example, height is used for vehicles as well as for persons, which mixes two different distributions).

To conduct the study, we collected all triples that use the three properties as predicates. The three datasets encompass 52,522 (`dbpedia-owl:height`), 206,997 (`dbpedia-owl:elevation`), and 237,700 (`dbpedia-owl:populationTotal`) triples. While it would be too expensive to build a complete gold standard for those datatsets, we only check whether the outliers identified by the different approaches are true or false errors.

We evaluate according to two dimensions: the outlier detection method itself (IQR, dispersion mode, KDE, iterative KDE, and KDE with FFT), as well as the preprocessing technique (*default*, i.e., no preprocessing, grouping by *single type*, as well as *clustering* by type vectors). For grouping by single types, we use only classes from the DBpedia ontology that represent leaves of the class hierarchy. Clustering by type vectors was done using the *FeGeLOD* framework with a threshold of 0.95 to create the type vectors, which are then clustered using the EM algorithm with a maximum of 100 iterations, no fixed number of clusters to create, and a minimum allowable standard deviation of 10^{-6} for normal density calculation. The reported results show the averages over all three predicates. For each of the algorithms, a large number of parameter settings was tested systematically.

The runtime for one analysis run in default mode on the three datasets were 1,832ms for IQR, 2,297ms for dispersion, 6,922ms for KDE-FFT, and 2,469,011ms (i.e., more than 40 minutes) for KDE. These runtimes include analysis overhead one the one hand and some caching on the other hand so they should only be viewed in comparison to each other, not as absolute values. In single type mode, IQR takes 41,538ms, KDE-FFT 31,336ms and dispersion 72,852ms for one sample run.

The clustering mode suffers from extremely high runtimes of around one hour for the height dataset, and over 24 hours for the larger `dbpedia-owl:elevation` and `dbpedia-owl:populationTotal` datasets. For those, about an hour is spent creating the vectors, using the public DBpedia SPARQL endpoint for retrieving the types, and the rest of the time running the actual clustering, which is the bottleneck of this approach.

The results of the pre-study are depicted in Fig. 3.1. As we are interested in methods for automatically detecting outliers, we were aiming at finding methods with high precision. Thus, we chose a set of parameters for each approach which optimizes the trade-off of precision and total number of outliers in a way that clearly prefers precision, such that if applied in an automatic setting, the probability of removing correct information is low. To that end, we use those parameters that optimize the trade-off between precision and absolute number of outliers. It can be observed that the precision for both dispersion and KDE FFT is much too low for the methods to be of actual use, since the loss induced by rounding to the next available example is very high on the dataset at hand.

Grouping is obviously useful as no method is able to achieve more than 30% precision in the baseline default mode. On the other hand, both IQR and KDE can yield precision scores of over 80%, if combined with some method of grouping. The results of grouping by single types and clustering by type vectors are comparable, which makes grouping by single type more preferable, considering the high runtime of the clustering approach. Furthermore, we find some improvement in precision for KDE by applying it iteratively to the same dataset.

Looking at the total number of outliers that each method can detect, we find that while IQR and KDE are able to achieve similar precision scores, KDE is able to detect much more outliers than IQR. Indeed, KDE iterative default detects 1622 incorrect values, most of which are incorrectly truncated values of 1.52 meters in the `dbpedia-owl:height` dataset (see Fig. 3, albeit at a low precision of 18%, as those values are too close to the valid range of body heights. However, such frequent anomalies can be used to detect errors in the DBpedia extraction code (see section 4).

3.2 Evaluation on Random Resources

Since the combination of IQR and grouping by single type provided some of the highest precision scores (88%) as well as runtimes that seemed suitable for large scale analysis, we chose to evaluate this combination further on a representative sample from DBpedia. To construct that sample, we used 50 random resources as

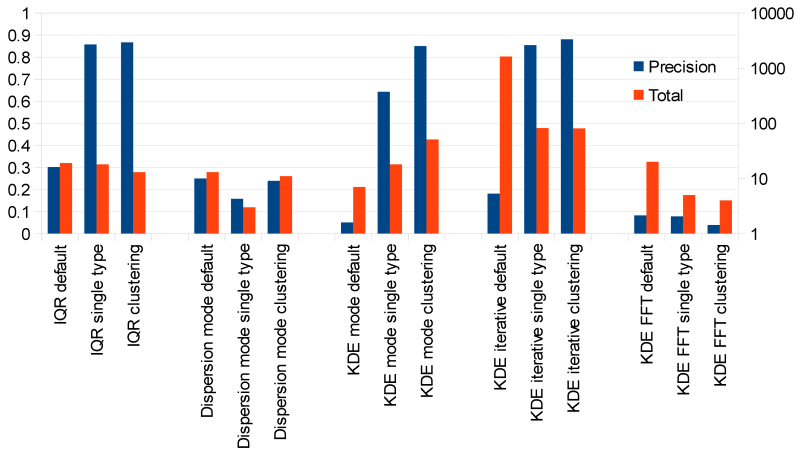

Fig. 1. Results of the pre-study on three selected properties. The x-axis shows the different approaches. Precision is shown on the left y-axis, the total number of outliers identified (true and false) is shown on the right y-axis.

a seed set, and collected all the data properties that have these resources as their subject. For those properties, we selected all triples that have these properties as their predicate, and used those for which more than 50% of all triples, and at least 100 in total, could be parsed to numeric values. This lead us to a random sample of 12,054,727 triples with literal, mostly numerical values.

As a first account, we used the parameters that had shown the highest precision in the pre-study (percentile=1.1, constant multiplier=50), and from that starting point, we systematically evaluated different parameter settings. The initial parameter setting yielded 1,703 suspicious triples, which we then evaluated manually. Manual verification was made feasible by using some shortcuts: The outliers did not occur at random but in clusters. For example, we found 122 area codes with eight or more digits. Since US area codes are all three digits in length, all triples with subjects of the form `[place],_[US state]` (which made up the vast majority) could be discarded at a glance. In some cases, however, we were not able to confidently determine what a certain property is supposed to represent and thus what values its objects should have. For example, the objects of the property `dbpprop:map` are so diverse that it is hard to tell what exactly they are supposed to represent. Thus, we labeled outliers for those properties as "unknown". Even if we pessimistically assume that all the outliers of unknown status are actually correct, IQR achieved a precision of 81% on our random sample. After removing the unclear data points, the highest precision is achieved at 88% with 859 true positives and 108 false positives, using a constant multiplier of 90 and 0.7 percentiles.

We also applied the methods in default mode to the sample data, which yielded nominally impressing results of thousands of outliers. The reason that we can find so many more outliers in general is that in order for an outlier to be found using the *single type* mode, its subject has to have a useful type, which is by far not given for all resources in DBpedia [12]. In our random sample, only two thirds of the subjects had at least one type.

Overall, three predicates, `dbpprop:date`, `dbpprop:years`, and `dbpprop:postalCode`, were responsible for the vast majority of those outliers. For those predicates, identifying true positives is easy because years and dates with more than four digits do not make sense, and the same holds for postcodes with more than ten digits.

By merely counting the corresponding objects for those three predicates, we would achieve true positive to unknown/false positive ratios of 7838:8751 (89%) with dispersion (MAD, constant factor = 300,000) and 5917:7216 (82%) with IQR (percentile = 1, constant multiplier = 41).

To verify that these results are not an effect of only a few low quality predicates, we chose to evaluate on the higher quality `dbpedia-owl` namespace only as well, which left us with 3,162,059 triples (26.2%) in 38 properties (22.6%) to analyze. Since we are dealing with a subset, the number of true and false positives cannot increase. With the same set of parameters, we find fewer outliers, but the overall trend remains, with 406 true positives (i.e., one wrong statement is identified in a thousand statements), and a precision of 87%.

4 Error Analysis

Based on the results obtained in our quality evaluations, we further examined common patterns in the errors we found to identify their causes. There are two basic classes of errors: those that exist as factual errors in Wikipedia, and those that occur while parsing the data from Wikipedia to DBpedia. Figure 2 shows the distribution of the error sources we identified.

In the following, we provide examples and explanations for errors found with our approach,[7] roughly classified into errors in Wikipedia, problems parsing primitive values, problems parsing non-primitive values, and problems interpreting and converting units.

4.1 Errors in Wikipedia

In some cases, the data is already wrong at the source, i.e. the Wikipedia page. For example, the page `http://en.wikipedia.org/wiki/Lerma,_State_of_Mex-`

[7] Note that since our study was performed on DBpedia 3.8, all examples shown refer to that version. Since, as a result of the research reported in this paper, we reported these errors to the DBpedia development team during our investigation, some of those have already been fixed for the latest DBpedia release, so not all of those errors can be reproduced with the latest version of DBpedia. Likewise, some of the examples for wrong data in Wikipedia used in this paper may have been fixed since this paper has been written.

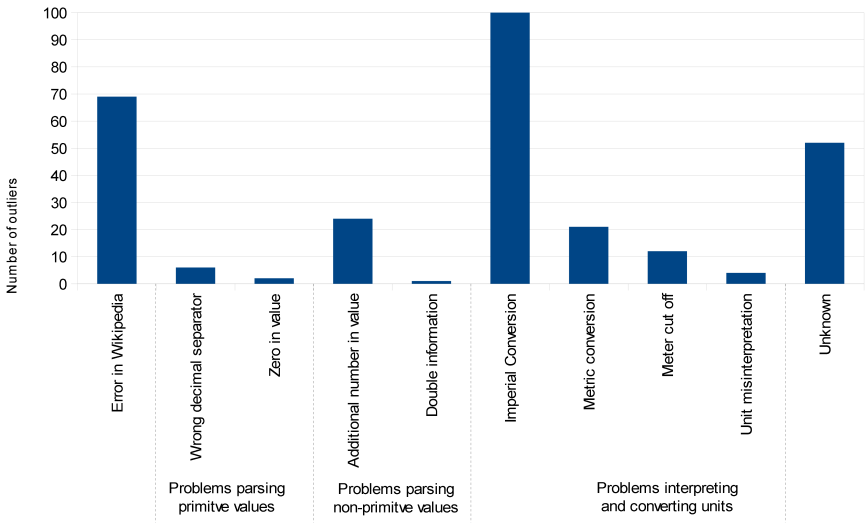

Fig. 2. Distribution of error sources. The value for *Imperial Conversion* is 1,506; the y-axis has been cut off at 100 for providing a better visualization.

ico gives the elevation of a town in Mexico as *25,700m 84,300ft*, which is clearly a wrong elevation, as it would be roughly three times higher than Mount Everest. These errors are hard to quantify, as they do not seem to follow a specific pattern.

In other cases, the correct data exists at Wikipedia along with another incorrect value, and the wrong value is selected during the extraction of DBpedia. For example, the page `http://en.wikipedia.org/wiki/Portland,_Michigan` gives the elevation of this town as *30,035ft (221m)*. 221 meters would be the correct value, but the DBpedia extraction code picks up the incorrect elevation in feet and converts it to the value of 9154.67 meters.

Some infobox keys in Wikipedia are used with inconsistent semantics. One example is the property `dbpprop:runtime`. Used for TV shows, it most often denotes the runtime of a single episode, while in some cases, it denotes the total time the series was aired. For example, the series *Wielie Wielie Walie*, a South African children's program, was aired for 18 years. The running time *18 years* is then transformed to a runtime of `568,036,800` seconds.

4.2 Problems Parsing Primitive Values

In this paper, we concentrate on numerical data. Such data can be obtained from primitive (simple numbers) as well as non-primitive (several numbers in one value, e.g., a population value and a year in which the population figures were collected) values. One problem in an earlier version of the DBpedia extraction

code was that some characters were converted to additional zeros in numbers. For example, `dbpedia:Durg` gives the population of the city of *Durg* as *2810436*, when it really is 281,436. A similar error can be seen with regards to the city of Nantong (`dbpedia:Nantong`). DBpedia gives the population as *72828350*, while, according to Wikipedia, it is actually 7,282,835.

Another class of problems comes from misinterpretation of wrong thousands and decimal separators.[8] For example, the comet *1134 Kepler* (`dbpedia:1134_Kepler`) has an eccentricity of *0.4650*, which is given in Wikipedia as *0,4650*, and gets misinterpreted to `4650` in DBpedia. Similarly, there are cases where dots are used as a thousands separator, e.g., for the city `dbpedia:Garg%C5%BEdai`, which has a population of *16,814*, defined as *16.814* in Wikipedia, which, after rounding, becomes a population of `17` in DBpedia.

4.3 Problems Parsing Non-Primitive Values

The most prominent indicator of non-primitive values is the presence of an additional number in the value. This happens, for example, with a year given for another value, such as a population. For example, the population of the village *Semaphore* (`dbpedia:Semaphore,_South_Australia`) is given as `28,322,006` (which exceeds the total population of Australia), when it is actually *2,832* – here, the year *2006* is reported next to the population, and the two numbers get concatenated during the extraction. A similar phenomenon can be observed for some runtimes, e.g., the runtime of the song *Last Christmas*, which is given as *3:02 (1946 recording)* in Wikipedia, and misinterpreted as `1,946` seconds.

A similar case is the presence of different numbers (e.g., ranges) for one property. This occurs, for example, with area codes, which are frequently given as lists or ranges, for example, Wikipedia gives the area code of *Central California* as *805, 559, 831*, which the extraction code turns into `805559831` at `dbpedia:Central_California`. Since Wikipedia entries can contain arbitrary text where a proper number may be expected by the DBpedia extraction code, entries such as *Mid-90s* may also be misinterpreted, leading to the starting date of `90 A.D.` for the band *Depswa* (`dbpedia:Depswa`).

In some cases, double information is extracted from more than one place. For example, the Wikipedia article for *Johnstown, Colorado* (`http://en.wikipedia.org/wiki/Johnstown,_Colorado`) gives its population as *9,887*. However, in the introduction, a population of *3,827* in 2000 is also mentioned. The two values get concatenated, so that in DBpedia (`dbpedia:Johnstown,_Colorado`), we find a combination of those two values as `38,279,887`.

4.4 Problems Interpreting and Converting Units

There are properties which use different units of measures. For example, the runtime of films (`dbpprop:runtime`) usually uses time intervals, such as hours,

[8] As we only look at the English language Wikipedia, these should in theory be free from regional variations.

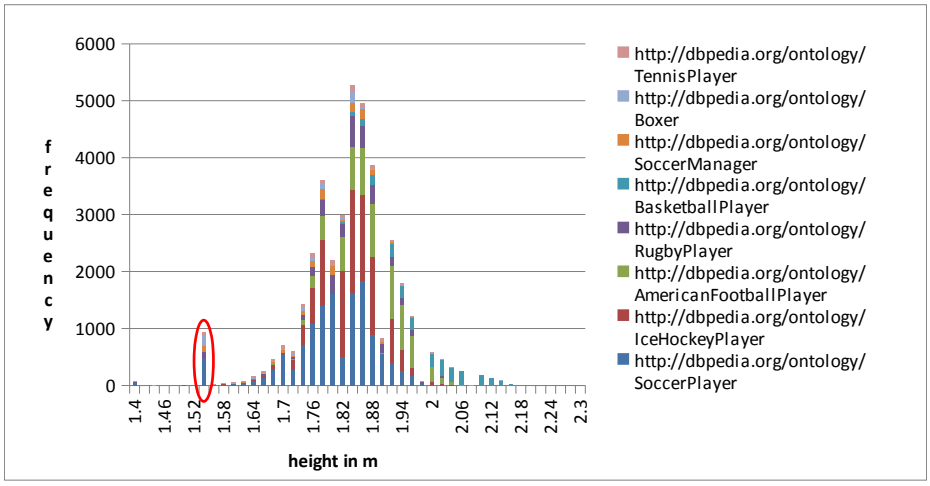

Fig. 3. Distribution of persons' heights, showing an anomaly at 1.52m.

minutes, and seconds, and is represented in DBpedia as seconds. However, there are also films such as the 1919 movie *The Unpardonable Sin*, whose runtime is given as *9 reels (2,700 meters)*, which is converted to `9` (i.e., nine seconds) in DBpedia (`dbpedia:The_Unpardonable_Sin_ (1919_film)`).

Another typical problem can occur with height or length values that are reported in mixed notation, using the unit of measurement in the middle. This leads to the *meter cut off problem*: a set of people with actual heights of around 1.5-1.9 meters have their heights represented in DBpedia as exactly 1.0 meters. This does seem to happen most often with somewhat unclean height specifications in Wikipedia. For example, Wikipedia states the height of the footballer *Guy Poitevin* as *1 m 81, 80 kg*, which then gets interpreted as 1.0m at `dbpedia:Guy_Poitevin`.

The by far largest source of errors happens during conversion of imperial units. In many cases, the given value differs only between about 0.025 and 0.3 meters from the actual value. In most cases the given height equals a round number of feet. For persons, the most common incorrect height is 1.524 meters or 5 feet; for example, the goalkeeper *Ray Wood* (`dbpedia:Ray_Wood`) is *1.80* meters in height according to Wikipedia, but DBpedia gives his height as `1.524` meters. This indicates that this error is caused by an incorrect parsing procedure from Imperial units to metric units where only the value in feet is correctly read while the remaining inches are cut off. Such errors can be observed as frequent anomalies in the distributions, as shown in Fig. 3.

The interpretation of values in metric units is also not free of errors. In some cases, heights appear too small by a factor of one hundred. For example, the correct height for `dbpedia:Humberto_Contreras` would be *1.76 meters* according to Wikipedia, however, DBpedia gives a value of `0.0176` meters, since the extraction code expects a value in centimeters, not meters. A similar error can

be observed with regards to some resources that have their height given in millimeters at Wikipedia, e.g. `http://en.wikipedia.org/wiki/FS_Class_E491/2` states the height of this locomotive as *4,310 mm (14 ft 1.7 in)*, which is rendered as `0.004310` [meters] at DBpedia. However, in other cases, the error is already present in Wikipedia. For example, the height of athlete *Katrina Porter* (`http://en.wikipedia.org/wiki/Katrina_Porter`) is given as `1.55cm`, which DBpedia correctly converts to `0.0155` meters.

It may also occur that imperial units are interpreted as metric, or vice versa. For example, Wikipedia gives the elevation of *Shadow Mountain Lake* as *8367'* (8367 ft.), which DBpedia misinterprets as `8,367` meters, thus causing the parsed value at `dbpedia:Shadow_Mountain_Lake` to be about three times (1 meter = 3.28084 feet) higher than it actually is. The entry on the *Zapatoca* mountain (`dbpedia:Zapatoca`) features even two of those errors: Wikipedia gives the elevation as *1,720 m (4,000 ft)*, and DBpedia renders this as both `1219.200000` and `13123.000000`. The first value corresponds to converting 4,000ft to meter and the latter to converting 4,000 meter to feet.

Time values are also prone to misinterpretation. Wikipedia lists the runtime of some albums as *mm:ss:msms*, for example `http://en.wikipedia.org/wiki/Les_Dudek_(album)`. The DBpedia extraction codes interprets this as *hh:mm:ss* and converts it to `2589.1666666666665` [minutes] at `dbpedia:Les_Dudek_(album)`.

5 Related Work

In [18], a taxonomy of errors in LOD is introduced. The taxonomy consists of four dimensions (accuracy, relevancy, representational consistency, and interlinking), seven categories, and 17 sub-categories. It encompasses plain errors, such as incorrectly extracted triples, as well as undesirable features, such as information being redundant or irrelevant. The errors found by the approach discussed in this paper mainly fall into the first category, i.e., incorrectly extracted triples.

The problem of automatically detecting errors in knowledge bases automatically has been acknowledged to be hard. In [16], an approach is evaluated of first enriching the DBpedia ontology with additional domain and range restrictions, as well as class disjointness axioms, and then using the enhanced ontology for error detection by reasoning. A similar approach is discussed in [8], but no quantitative results on DBpedia are provided. However, these two approaches target at finding wrong statements involving object properties, i.e., relations between two resources, rather than wrong numerical literals.

Other approaches use external knowledge to validate statements, either from experts or from external data sources, and with different scopes (e.g., validating both object and data type properties vs. only data type properties). [1] discuss crowd-sourcing, using platforms such as Amazon Mechanical Turk which pay users for micro-tasks, such as the validation of a statement. Furthermore, they used a custom platform which organized the validation of statements as a competition. Their evaluation concentrates on three error classes, i.e., wrong literal

values, wrong literal datatypes, and wrong interlinks to other datasets. For the first, which we also address with our approach, they report a precision of 0.90, which is close to our results, which, however, does not rely on the wisdom of the crowds. [17] use games with a purpose to evaluate DBpedia and spot inconcistencies. They report that in 4,051 statements used in the game, 265 inconsistencies have been detected by users, 121 out of which were actually inconsistencies. This leads to a precision of only 0.46, which makes that approach only partially suitable for increasing data quality, at least without expert reviewing.

External knowledge is used, e.g., by DeFacto [9]. The authors have build a pattern library of lexical forms for properties frequently used in DBpedia. Using those lexical patterns, DeFacto runs search engine requests for natural language representations of DBpedia statements. While DeFacto seems to work on ObjectProperties, not DatatypeProperties, the approach is transferable to the problem of identifying errors in numerical data as well. Their approach reaches a precision of 0.88, which is comparable to our approach.

Overall, there are not too many approaches for automatically identifying wrong numerical values in Linked Open Data, in particular not without using external sources of knowledge, such as the wisdom of the crowds. Furthermore, it is interesting to see that even approaches using external knowledge sources do not reach significantly higher precision figures.

6 Conclusion and Outlook

In this paper, we have examined the possible usage of different outlier detection methods, i.e., Interquantile Range (IQR), Kernel Density Estimation (KDE), and dispersion estimators, for identifying wrong statements in DBpedia. The outlier detection methods are combined with different preprocessing strategies, i.e., grouping subjects by the single types, as well as clustering by type vectors. The simple IQR method delivers some of the best results in our tests. Combined with grouping by single type preprocessing, we achieved a precision of 87% on small high quality samples as well as on large random samples. Basic KDE shows similar results, but suffers from high runtimes. The other methods examined, i.e., dispersion and KDE-FFT, mostly fail to deliver results with sufficient precision.

The evaluation has shown that exploiting further semantics in DBpedia, i.e., the type information of the statements' subjects, leads to an improvement compared to simply applying outlier detection to all numerical values of a property at once. Clustering by type vectors does produce promising results as well but is, at least in our current implementation, not feasible runtime-wise. Iterative application of analysis methods does not improve results for most methods. Only KDE clearly benefits from using more than one iteration; with all other methods, results either do not change or, if they were bad to begin with, tend to get even worse. Overall, the combinations of IQR and iterative KDE, and grouping by single type or clustering by type vector, produce the best results.

As a result of applying our approach, we identified a number of common sources of errors in DBpedia. Large amounts of the faulty numerical values in

DBpedia are caused by only a few of those error sources. Overall, 11 different types of errors in the DBpedia extraction framework regarding properties in the `dbpedia-owl` namespace were identified and forwarded as bug reports to the developers of DBpedia, many of which have been resolved for the current DBpedia release. While we were identifying these errors by manual inspection, automatically detecting patterns for data formats that are not handled correctly would be a useful extension of the approach.

So far, we have only considered numerical data, i.e., integer or double values. Extending the approach to dates would be interesting, straight forward, and particularly useful, since dates, like numbers, are prone to being parsed wrongly.

While the clustering approach showed promising results, but had runtime problems, there is clearly room for improvement here. We evaluated one clustering approach on RDF types, which showed promising results but suffered from extremely high runtimes. However, there is an abundance of clustering algorithms, and there is much more information available for each subject beyond its type that could be used in the clustering process, e.g., vectors of relations or Wikipedia categories. Other feature vector representations, combined with different clustering algorithms, could improve the clustering results as well as runtime.

In general, outlier detection as a method of identifying errors has some fundamental limitations in that in order for an erroneous data point to be detected, it has to be "outlying" in some numerical manner. For example, if all regular ZIP codes have five digits, it should be possible to detect invalid ZIP codes with 3 or 14 digits. However, if a ZIP code is simply wrong as in *96377* instead of *94303*, it will be practically impossible to detect as an outlier without using background knowledge.

In this paper, we have restricted ourselves to applying outlier detection methods on single attributes. While the results are promising, using more than one variable at the same time seems promising, e.g., finding outliers in cities' *populations* by taking the *area* attribute into account (assuming that cities with larger population also occupy a larger area).

So far, we have only considered one particular data source, i.e., DBpedia. While the approach itself is transferable to any RDF knowledge base, exploiting links to other datasets and using information from more than one dataset at the same time could help further improving the results. By comparing suspicious values to corresponding values from other sources, e.g., other language editions of DBpedia, it could not only be possible to detect more outliers, but also to correct them automatically too. Applying the findings not only on DBpedia, but also on Wikipedia directly, e.g., in the form of editing support, would be another possible application.

In summary, we have shown that even basic outlier detection methods, combined with suitable preprocessing strategies, lead to highly effective error detection mechanisms.

References

1. Maribel Acosta, Amrapali Zaveri, Elena Simperl, Dimitris Kontokostas, Sören Auer, and Jens Lehmann. Crowdsourcing linked data quality assessment. In *International Semantic Web Conference (ISWC)*, 2013.
2. Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, 5(3):1–22, 2009.
3. Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977.
4. Frank E Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.
5. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: an update. *SIGKDD Explor. Newsl.*, 11:10–18, November 2009.
6. Wolfgang Hardle. *Nonparametric and semiparametric models*. Springer Verlag, 2004.
7. Victoria J Hodge and Jim Austin. A survey of outlier detection methodologies. *Artificial Intelligence Review*, 22(2):85–126, 2004.
8. Jens Lehmann and Lorenz Bühmann. Ore-a tool for repairing and enriching knowledge bases. In *The Semantic Web–ISWC 2010*, pages 177–193. Springer, 2010.
9. Jens Lehmann, Daniel Gerber, Mohamed Morsey, and Axel-Cyrille Ngonga Ngomo. Defacto-deep fact validation. In *The Semantic Web–ISWC 2012*, pages 312–327. Springer, 2012.
10. Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. DBpedia – A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. *Semantic Web Journal*, 2013.
11. Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.
12. Heiko Paulheim and Christian Bizer. Type inference on noisy rdf data. In *12th International Semantic Web Conference (ISWC)*, 2013.
13. Heiko Paulheim and Johannes Fürnkranz. Unsupervised Generation of Data Mining Features from Linked Open Data. In *International Conference on Web Intelligence, Mining, and Semantics (WIMS'12)*, 2012.
14. Bernard W Silverman. *Density estimation for statistics and data analysis*, volume 26. CRC press, 1986.
15. BW Silverman. Algorithm as 176: Kernel density estimation using the fast fourier transform. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 31(1):93–99, 1982.
16. Gerald Töpper, Magnus Knuth, and Harald Sack. Dbpedia ontology enrichment for inconsistency detection. In *Proceedings of the 8th International Conference on Semantic Systems*, pages 33–40. ACM, 2012.
17. Jörg Waitelonis, Nadine Ludwig, Magnus Knuth, and Harald Sack. Whoknows? evaluating linked data heuristics with a quiz that cleans up dbpedia. *Interactive Technology and Smart Education*, 8(4):236–248, 2011.
18. Amrapali Zaveri, Dimitris Kontokostas, Mohamed A Sherif, Lorenz Bühmann, Mohamed Morsey, Sören Auer, and Jens Lehmann. User-driven quality evaluation of dbpedia. In *9th International Conference on Semantic Systems (I-SEMANTICS '13)*, 2013.