Divided we stand out! Forging Cohorts fOr Numeric Outlier Detection in large scale knowledge graphs (CONOD)

Hajira Jabeen[1], Rajjat Dadwal[2], Gezim Sejdiu[1], and Jens Lehmann[1,2]

[1] University of Bonn, Germany `lastname@cs.uni-bonn.de`
`https://www.uni-bonn.de/`
[2] Fraunhofer IAIS, Germany - {`firstname.lastname`}`@iais.fraunhofer.de`

Abstract. With the recent advances in data integration and the concept of data lakes, massive pools of heterogeneous data are being curated as Knowledge Graphs (KGs). In addition to data collection, it is of utmost importance to gain meaningful insights from this composite data. However, given the graph-like representation, the multimodal nature, and large size of data, most of the traditional analytic approaches are no longer directly applicable. The traditional approaches could collect all values of a particular attribute, e.g. height, and try to perform anomaly detection for this attribute. However, it is conceptually inaccurate to compare one attribute representing different entities, e.g. the height of buildings against the height of animals. Therefore, there is a strong need to develop fundamentally new approaches for the outlier detection in KGs. In this paper, we present a scalable approach, dubbed CONOD, that can deal with multimodal data and performs adaptive outlier detection against the cohorts of classes they represent, where a cohort is a set of classes that are similar based on a set of selected properties. We have tested the scalability of CONOD on KGs of different sizes, assessed the outliers using different inspection methods and achieved promising results.

Keywords: Knowledge Graph · Cluster · Outlier · Blocking · Cohort · RDF · DBpedia .

1 Introduction

Numeric outlier detection has been of interest for the database and data mining community for efficient and automatic detection of errors, frauds, and abnormal patterns. A range of algorithms have been devised for outlier detection of homogeneous set of features like in traditional Databases or tabular data. However, with the current influx of large-scale heterogeneous data in the form of knowledge graphs (KGs), most of the traditional algorithms are no longer directly (out-of-box) applicable and there is a strong need to develop fundamentally new approaches to deal with the volume and variety of large-scale KGs.

A KG defines classes and relations among these classes in a schema, allows linking of arbitrary entities (instances of classes) with defined relationships and covers a variety of domains. A multitude of approaches are being used to populate and curate KGs. These range from crowdsourcing (DBpedia), application of natural language processing techniques (NELL) to automatic extraction of knowledge (YAGO, DBpedia). The liberal nature of these curation methods – the knowledge being entered is neither restricted nor cross-validated – makes them prone to various kinds of errors that can get camouflaged in different dimensions. These errors can be extraction errors like parsing errors, e.g. "3-4" can be interpreted as "3", and "-4" , representation errors and conversion errors, e.g. the units of measurements cm, inches, feet are mixed, or data entry errors.

Numeric data can be associated with entities belonging to numerous classes and one property can be associated with several different types, e.g. the property "height" can be associated to a building, an animal, or a person. It is of utmost importance to treat a particular property in correspondence with the type of entities it represents when performing outlier detection. In this paper, we have focused on outlier detection on numeric literals within a KG.

Anomaly or outlier detection is mostly regarded as an unsupervised learning task dealing with identifying unlikely and rare events. Outliers are usually the extreme values that deviate from the remaining observations. It is challenging to devise a scalable yet generic method that can detect outliers across different dimensions automatically without manual intervention. In this paper, we have targeted the above challenges of detection of outliers in multimodal and multivariate data with different distributions coming from disparate sources curated as a large KG. We propose an unsupervised numeric outlier detection method that is generic and yet scales to large KGs. Our approach 'Forging Cohorts for Numeric Outlier Detection in large scale KGs (CONOD)' proceeds in two stages. In the first stage, the data is linearly cohorted using the type information of the entities. In the second stage, the outlier detection is performed on common numerical literal properties within each cohort. The major contributions of CONOD are 1) Scalability, 2) Generic-ness 3) Linear time cohorting, 4) Applicability to multimodal data. We have tested the scalability of CONOD on DBpedia and its scaled-out versions. To the best of our knowledge, CONOD is the first open source, generic method for unsupervised numeric outlier detection on KGs.

The rest of the paper is structured as follows: Section 2 discusses the preliminaries, Section 3 reviews the related work and identifies the existing research gaps, followed by the proposed approach, Experiments and evaluation, and Conclusions and future work in Sections 4, 5, and 6 respectively.

2 Preliminaries

2.1 Semantic web and DBpedia

The rationale behind the semantic web is to represent information by providing machine and human understandable descriptions of real world things (resources).

The Resource Description Framework (RDF) is a W3C standard model for representing semantic relationships between data items. The RDF data is represented as a set of triples containing a subject, a predicate, and an object. Subjects are the resources represented by unique URIs, objects can either be a resource or a literal. The predicate is the relationship between the subject and the object. The membership of a resource to its classes is defined by the *rdf:type* property and each resource can belong to many classes.

DBpedia [7] is a community effort to extract structured information from Wikipedia and to make this information available on the Web in the form of linked data[3]. DBpedia released its first dataset in 2007 by the developers of University of Mannheim and University of Leipzig. Wikipedia consists of information like images, numerical attributes, e.g. population or height, links to external web pages and structural information. In its current version, DBpedia contains information about 4.58 million things, including 1,445,000 persons, 735,000 places (including 478,000 populated places), 87,000 movies among many others.

2.2 Anomaly detection

Outlier or anomaly detection is a technique for "finding patterns in data that do not conform to the expected normal behavior" [1]. A broad range of outlier detection methods have been proposed in the literature and they can be roughly categorized as being based on nearest neighbors, clusters, or metrics like density, distance, depth, and statistics for outlier detection. In this work, the focus is to apply a simple outlier detection method for univariate numerical outliers. Therefore, we discuss a few prominent, univariate, statistical methods for outlier detection.

Grubbs' method The Grubbs' method [5] also known as the maximum normed residual test is used to detect a single outlier in a univariate data set that follows an approximately normal distribution. The Grubbs' test is defined as: $G = \frac{\max |Y_i - \bar{Y}|}{s}$, with $\bar{Y}$ and s denoting the sample mean and standard deviation, respectively. The Grubbs' test statistic is the largest absolute deviation from the sample mean in units of the sample standard deviation. The Grubbs' procedure tests the hypothesis whether the value that is the furthest from the uniform sample mean is an outlier. Therefore, it is not suited for data that is non uniformly distributed or contains blocks of outliers.

Inter Quartile Range The Inter Quartile Range IQR method [10] is the amount of spread in the middle 50% of a dataset. It is the distance between the first quartile $Q1$, Median $M($ or $Q2)$ and third quartile $Q3$ of the given numerical dataset. Quartiles divide a rank-ordered data set into four equal parts. The values that separate parts are called the first, second, and third quartiles; and they are denoted by $Q1$, $Q2$, and $Q3$, respectively.
$Q1$ is the median of all the values smaller than the median M whereas the $Q3$ is the median of all the values higher than the median M. IQR is the measure of variability defined by the difference between $Q3$ and $Q1$. The data points which

[3] `https://en.wikipedia.org/wiki/DBpedia`

are smaller than $Q1 - 1.5 * IQR$ and greater than $Q3 + 1.5 * IQR$ are considered as outliers. The constant value 1.5 depends upon the distribution of the data and can be adjusted accordingly.

Median Absolute Deviation Median Absolute Deviation (MAD) [9] measures the variability of a univariate sample of quantitative data in statistics. It is more resilient to outliers in a data set than the standard deviation method. MAD is defined as the median of the absolute deviations from the data's median for univariate data. The median absolute deviation is defined as :
$MAD = b * median\,(\ |X_i -\ median(X)|\ )$
Outlier detection using MAD can be performed by calculating $A = Median + 2.5 * MAD$, and $B = Median - 2.5 * MAD$. Any value in the input set X which is greater than A and less than B is considered an outlier.

Leys et. al. [9] have surveyed different methods of outlier detection. They argue that the 'mean' of any data have zero breakdown point(One infinite value in data shifts the mean to infinite) that makes it unsuitable for the calculation of outliers. Whereas Median breakdown value is about 50%, meaning that the median can resist up to 50% of outliers. MAD also has the same breaking point, whereas IQR has a breakdown point of 25%. Based on the resilience, we have chosen MAD and IQR to find the outliers in RDF data.

2.3 Apache spark for big data processing

Apache Spark is a scalable, in-memory, general-purpose cluster computing framework with APIs available in Java, Python, and Scala[4]. Apache Spark, consolidated under one stack, consists of several independent special purpose libraries.

1. **Spark Core** includes basic functionalities of spark like task scheduling, memory management, fault recovery and interaction with the storage system. It is also home to the Resilient Distributed Dataset(RDD) API.
2. **Spark SQL** is the component of the Spark stack that deals with structured data by providing the DataFrames API. The basic SQL operations can be performed on DataFrames.
3. The **Cluster Manager** is designed for distributed computing where parallel operations run on various computer nodes. Spark can use its inbuilt standalone scheduler or can use other cluster managers like Apache Mesos or Hadoop yarn.

Resilient Distributed Dataset (RDD) RDD [15] is an immutable and distributed scala collection. In Spark, everything is expressed as RDD. An RDD can be created by:

1. Loading an external dataset in Spark,
2. Parallelizing an existing collection of objects,
3. Manipulating existing RDDs.

Two types of operations can be performed on RDDs:

[4] `https://spark.apache.org/docs/latest/`

1. Transformation: Return another RDD after applying a function on existing RDDs This is done via a lazy execution, i.e. the result is not immediately computed.
2. Action: Compute a result based on an RDD and either return or save it to an external storage system. Here, the computation is eager, i.e. the result is immediately computed.

The lazy evaluation of Spark helps in achieving fault tolerance and optimizing performance. Spark creates a lineage graph for all transformations and executes it optimally when an action is triggered. This lineage graph is used to recover lost computing information upon a node failure. Spark offers a range of inbuilt functions for RDD operations executing in parallel with little to no programming overhead.

3 Related work

The literature on outlier detection in KGs is relatively scarce because KGs had mainly been considered for data consolidation or curation and their potential for analytics has gained momentum in the recent years only. Below, we discuss a few existing outlier detection methods for KGs.

Weinand et al. [14] presented an experimental approach for the detection of numeric outliers from DBpedia. The authors argued that the traditional outlier detection approaches are limited by the existence of natural outliers. Therefore, they group similar objects together and then apply outlier detection exclusively on these grouped objects to overcome this problem. The grouping incurs high complexity as it uses the type information, which is not always present or is either too generic (e.g. owl:Thing as only type) or is inaccurate. Due to being computationally expensive, the proposed method was tested on only part of DBpedia using the SPARQL endpoint for three numerical properties (*DBpedia-owl:populationTotal*, *DBpedia-owl:height*, and *DBpedia-owl:elevation*). Additionally, the authors have compared different outlier detection techniques, i.e., Inter Quartile Range (IQR), Kernel Density Estimators (KDE) [12] and dispersion estimators and reported that the IQR performs better than other methods tested. In a similar approach [13] Paulheim used hot encoding for type vectors of entities and clustered them into groups before performing anomaly detection on numerical features for identification of wrong links in the data.

Fleischhacker et al. [4] presented an outlier detection method that crosschecks the results of outliers by exploiting the "*sameAs*" properties in the knowledge base and also makes an effort towards differentiating between natural outliers and actual outliers in the data. The outlier detection method is carried out by dataset inspection through specialized SPARQL queries against the knowledge base. In the first step the authors select the interesting properties for outlier detection. In the second step, the subpopulation is sought by using specialized operators for developing a set of constraints (top-down ILP algorithms for discovery of datalog rules) against classes, properties, and property values. This exploration is organized as a lattice where the root node consists of a property and the corresponding number of instances. After the lattice has been determined, the next step is to find the outliers on all unpruned nodes of the lattice. The results of the outlier score are stored as a set of constraints which returns the corresponding instance set. The classification of outliers into natural or real is done with the help of data interlinking property by comparing with different datasets. This procedure helps in better handling natural outliers and thus reducing the false positive rate. However, building the lattice for the subpopulation discovery requires substantial memory and computation [11]. In addition, the method requires manual querying of data in order to extract the required information. Therefore, the method is unsuitable for very large scale knowledge bases.

Debattista et al. [3] have presented a preliminary approach for distance-based outlier detection for linked data quality improvement. They detect incorrect RDF statements by applying a distance-based clustering method for pointing out the outliers in linked data.

In summary, the outlier detection methods discussed above have the limitations of:
a) Accessing the data through a SPARQL endpoint(slow, unreliable),
b) Using Clustering(mostly with quadratic complexity),
c) Using Sub-latex search(complex),
and therefore are not scalable to large-scale knowledge graphs, which is the main contribution of this work.

4 Approach

In this section, we detail **CONOD**, a scalable and generic algorithm for numeric outlier detection in KGs. In order to deal with the multimodality of the data,

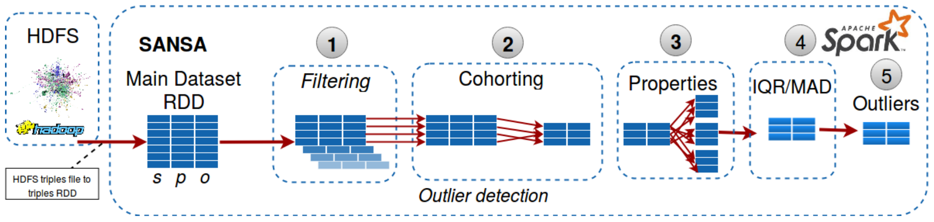

Fig. 1: CONOD execution pipeline

we have used a cohort-based approach. A cohort can be defined as the set of classes that share some similarities based on their selected features. CONOD can be divided into following steps 1) data cleaning, 2) cohort creation, 3) property selection, 4) outlier detection(IQR) and 5) output, as shown in Figure 1. Algorithm 1 describes the working of CONOD that takes RDF triples as input, creates RDD, applies a series of transformations and actions, and returns a list of outliers as output.

Algorithm 1: CONOD

```
Result: Outliers list
input : $RDF$: an RDF dataset
1  $RDD\ mainDataset = RDF.toRDD < Triple > ()$
2  $mainDataset.cache()$
3  $Numerics \leftarrow filter(mainDataset)$
4  $Subjects \leftarrow filter(mainDataset)$
5  $FilteredData \leftarrow Numerics \cup Subjects$
6  $PairedRDD \leftarrow FilteredData.GroupBy(Subjects)$
7  $VectorizedData \leftarrow Vectorize(PairedRDD)$
8  $Hashes \leftarrow LSHASH(VectorizedData)$
9  $Cohorts \leftarrow similarityByJoin(Hashes, Hashes, threshold)$
10 $Properties \leftarrow Cohorts.getPProperties()$
11 $Outliers \leftarrow IQROutlier(Properties)$
```

4.1 Data cleaning and filtering

KGs are created by collecting data from different sources represented in the form of triples. These triples can contain relationships of numerous types. As we are interested in numerical values for the outlier detection, we use a data filtering step to extract the data of interest. Here, we also assume that the data contains a schema and has the information about the rdf:type for most of the resources and is possibly enriched with Linked data Hypernyms [6].

The Linked Hypernyms Dataset (LHD) provides types in DBpedia namespace. These types are extracted from the first sentences of Wikipedia articles from different languages using Hearst pattern matching over part-of-speech annotated text and disambiguated to DBpedia concepts. The cleaning step selects the triples with objects having literals of type "xsd:integer, xsd:nonNegativeInteger and xsd:double". At a later stage, for cohorting, we use the information about type and LHD of the subjects of filtered triples. Therefore, we also filter the type information at this stage. This filtering is done in step 1-5 of Algorithm 1.

4.2 Creation of cohorts

In this step, the filtered data is used for creation of cohorts by using Locality Sensitive Hashing(LSH) [2]. LSH is an important class of hashing techniques. LSH uses a family of functions ("LSH families") to hash data points into buckets, so that the data points which are close to each other are in the same buckets with high probability, while data points that are far away from each other are likely in different buckets. As discussed in the previous sections most of the outlier detection methods use some sort of clustering to find the outliers in the data which are quadratic in nature. The proposed LSH based cohorting achieves scalable performance using linear complexity. The process of cohorting can be described as:

1. Select the type information and LHD types of each subject and represent them as a key-value pair, with subject as the key and type-and-LHD information as the values.
2. Convert this type information into vectors named featureVectors
3. Create hashes of the featureVectors using LSH hashing.
4. Find similarity between hash-vectors and create cohorts
5. Output cohorts

We have experimented with two different vectorizing models CountVectorizer and HashingTF. We have used similarity join method for measuring the similarity among different values. This can be seen in step 6-9 of Algorithm 1.

4.3 Properties in cohorts

Once the cohorts of subjects are created, we group the numerical properties of these subjects within each cohort and perform outlier detection for each property in parallel. This is step 10 in Algorithm 1.

4.4 Outlier detection, and Output

Algorithm 1 uses the IQR method to find outliers from the group of properties in Step 11. The implementation of IQR is described in Algorithm 2. The IQR algorithm takes numeric properties as input, prepares a list of numbers corresponding to each property, applies the IQR method on this list and returns the list of outliers. The list of outliers is saved in HDFS for analysis purposes, e.g. classification of outliers, outlier analysis etc.

```
Algorithm 2: IQROutlier
  Result: Outliers list
  input  : numericProperties: Group of properties from each cohort
1 $RDD\ numericRDD = numericProperties.toRDD < Triple > ()$
2 $listNumerics \leftarrow filter(numericRDD)$
3 $sortedList \leftarrow sort(listNumerics)$
4 $Q_1 \leftarrow firstQuartile(sortedList)$
5 $Q_3 \leftarrow thirdQuartile(sortedList)$
6 $IQR \leftarrow Q_3 - Q_1$
7 $lowerRange \leftarrow Q_1 - 1.5 * IQR$
8 $upperRange \leftarrow Q_3 + 1.5 * IQR$
9 $outliersList \leftarrow$
    $filter(listNumerics < lowerRange\ or\ listNumerics > upperRange)$
```

4.5 Implementation

We have used the distributed in-memory computing framework *Apache Spark* to support horizontal scalability. CONOD has been implemented as a module in SANSA [8], an open source [5] *data flow processing engine* for performing distributed computation over large-scale RDF datasets. It provides data distribution, communication, and fault tolerance for manipulating massive RDF graphs and applying machine learning algorithms on the data at scale. We have used the SANSA-RDF layer for the ingestion of RDF data and its representation as RDDs. The algorithm is provided as an API in the machine learning layer of SANSA.

5 Experiment and evaluation

We have tested the performance of CONOD on a cluster with 4 servers having a total of 256 cores, and each server has Xeon Intel CPUs at 2.3GHz, 256GB of RAM and 400GB of disk space, running Ubuntu 16.04.3 LTS (Xenial) and connected via a Gigabit Ethernet2 network.

We have used Spark Standalone mode with Spark version 2.2.1 and Scala with version 2.11.11. Each Spark executor is assigned a memory of 250GB.

Table 1 provides an overview of the results generated by CONOD. Out of 117544372 triples, 22375991 numeric literals were selected for anomaly detection after the filtering process. These triples belong to 1567 distinct properties. From these properties only 408 were found to have outliers and the total number of outliers found in DBpedia are 24015, which is approximately 0.1 % of numeric literals present in the data.

Table 1: Statistics for DBpedia large file (16.6 GB)

Statistics	**Value**
Distinct Properties	2863
Triples (including duplicates)	117,544,372
Numeric literals after filtering process (including duplicates)	22,375,991
Filtered distinct numerical properties	1,567
The number of properties with outliers	408
Total number of outliers	24,015
– Runtime TV series has total number of outliers	482
– Built-year property of buildings has total number of outliers	86
– PostalCode of area has total number of outliers	176

[5] `https://github.com/SANSA-Stack`

5.1 Execution time

In this section, we discuss the scalability and runtime performance of CONOD. The runtime does not include the time for data ingestion from Hadoop file system.

Datasets and execution time In order to test the scalability, applicability, and efficiency of CONOD in terms of execution time, we have tested the approach on different sizes of DBpedia as shown in table 2.

Dataset Categorization	Dataset Size
DBpedia-Small	110 MB
DBpedia-Medium	3.7 GB
DBpedia	16.6 GB
DBpedia *2	32 GB
DBpedia * 4	64 GB
DBpedia * 8	128 GB

Table 2: Dataset Description

The execution times of CONOD on different datasets are shown in Figure 2. We can see the the run time increased with the increase of data. For DBpedia-Small it took only 2 minutes while for DBpedia * 8, it took approximately 5.9 hours. This can be noted that the time taken by CONOD increases by a factor of 1.5 when we double the size of data depicting the linear increase in runtime corresponding to the data size.

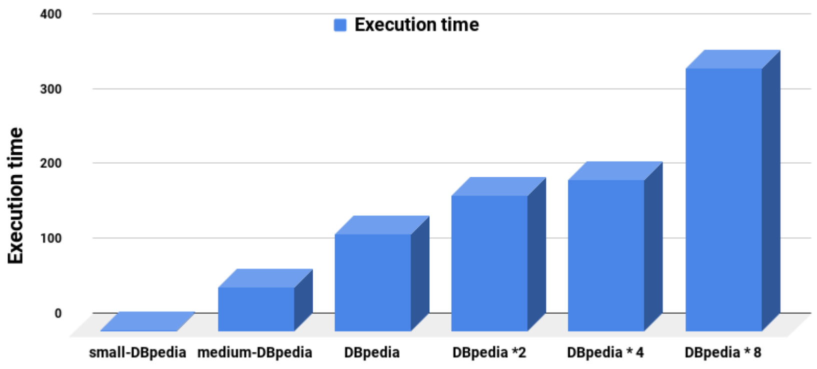

Fig. 2: CONOD: Execution time of different datasets

Comparison with other approaches In order to show the scalability and comparison with traditional methods, we have implemented a Cartesian product based similarity method for cohorting similar subjects (by using DataFrame's crossJoin function in Spark). This has also helped us to evaluate the speedup gained with CONOD. Here, we compare the runtime for both approaches, i.e., cohorting of classes by using the Cartesian product and CONOD (using two different vectorizing methods namely, CountVectorizerModel and HashingTF). In Figure 3, it can be noted that the execution time of the small dataset is almost equal for all the three methods because the small dataset does not require more resources like memory or execution cores.

For the medium-DBpedia, the DataFrame crossJoin function takes less time compared to other approaches. The other two approaches take more time due to approximate similarity join function that calculates the approximate similarity for finding the similar items in our algorithm and also the process of computing the vector from strings and creation of vocabulary for the vectorizing is noticeable for the medium sized data. Due to this reason, the execution time of HahingTF and CountVectorizerModel model is more as compared to crossJoin on medium datasets However, on DBpedia dataset, The DataFrame crossJoin approach fails on the cluster being tested due to its $O(n^2)$ complexity and we have represented this with three dots in figure 3. The CountVectorizerModel approach takes more time than the HashingTF on the large dataset as it scans the data twice, first time for building the model, and second time for the transformation. CountVectorizerModel needs extra memory equal to the number of unique features whereas HashingTF does not require additional space. By examining the results shown in the figure, it can be inferred that CONOD(HashingTF) performs better than the other approaches explored in this paper for larger datasets.

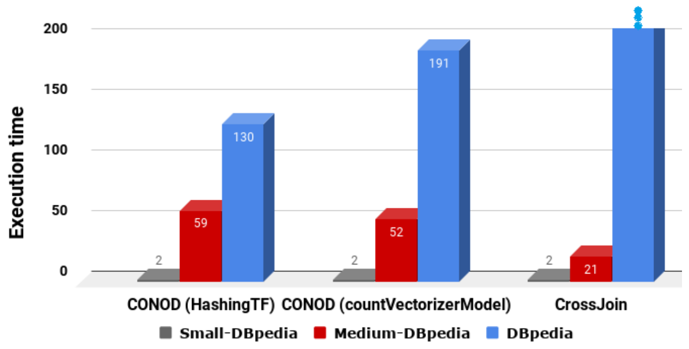

Fig. 3: Execution time comparison of different approaches

5.2 Visualization of outliers

Given a large number of outliers detected against a number of different properties, it is not straightforward to assess the quality of outliers. Therefore, we have created two dimensional scatter plots of the numerical values of a few randomly selected properties in order to visualize the outliers and analyze them.

We have used python matplotlib[6] library for creating the graphs. The red color in the graph corresponds to the outliers, whereas green represents the normal data values.

Built(year) property of buildings Built property represents the year when a building is built. Listing 1.1 shows that Foolad Shahr Stadium was built in 1988.

```
dbr:Foolad_Shahr_Stadium     dbo:built      1998^^xsd:integer.
 dbr:Ivaylo_Stadium          dbo:built      1958^^xsd:integer.
```

Listing 1.1: Built property in DBpedia

The Figure 4 shows the scatter plot of built property. The x-axis represents the years and the y-axis represents the frequency of occurrence of each year. Since built property signifies the past events, any value greater than the current year can be considered as a real outlier. e.g. a value in the x-axis shows that year has value around 0.9 x $1e^{19}$ and this is a outlier. The graph also shows an erroneous value that Metropolitan Life Insurance Company Hall of Records building was built in the year 9223372036854775807. One can observe in the zoomed-in graph, that the values near zero are shown as outliers in the plot.

Width property of cars We have plotted the outlier graph for the width of automobiles in Figure 5. The width property of cars has range xsd:double and xsd:integer. The Listing 1.2 shows that the automobile named as Toyota Avalon has width 1997 in DBpedia.

```
dbr:Jaguar_S−Type      dbo:width      "2007"^^xsd:integer.
dbr:Toyota_Avalon      dbo:width      "1997"^^xsd:integer.
```

Listing 1.2: RDF data of width property of cars present in DBpedia

The value around 2000 in the graph is an outlier because DBpedia extraction tool has extracted year in place of width.

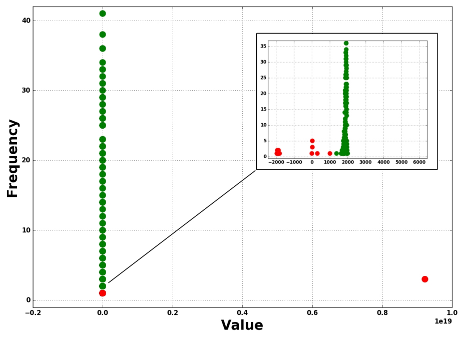

Fig. 4: Outliers 'Built-year'

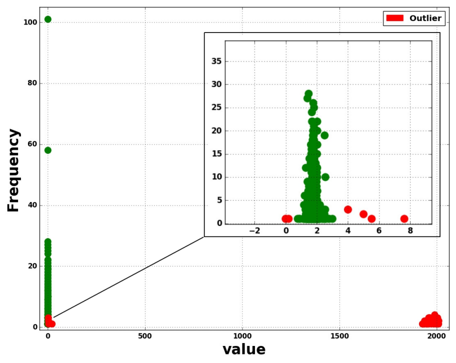

Fig. 5: Outliers 'Width of cars'

[6] https://matplotlib.org/

The zoomed-in graph in Figure 5 shows that most of the cars have width range from 0.627 to 3.011 and the values out of this range are outliers.

5.3 Manual inspection of outliers

In the manual inspection of outliers, we have compared the outliers with the corresponding Wikipedia pages to observe the reason. Given the large size of the knowledge base and a large number of outliers detected, the manual inspection of outliers of all the values is not feasible.

```
dbr:GS&WR_Class_201   dbo:builddate   188718951901^^xsd:integer.
dbr:GE_UM12C          dbo:builddate   19631966^^xsd:integer.
```

Listing 1.3: Example of real outliers in Dbpedia

Therefore, we randomly sampled some values for inspection. We have found some unusual patterns in the output as shown in the Listing 1.3 and classified them as real outliers.

The inspection also enabled us to figure-out the reasons behind these outliers in DBpedia.

Erroneous information in Wikipedia Given that Wikipedia is being maintained and updated by the community, and input to the pages is not strictly validated, Wikipedia is prone to containing erroneous information. This incorrect information results in the outliers in DBpedia. The correctness of these outliers is difficult to assess as it needs to be confirmed from the external sources or to be validated by experts.

DBpedia extraction tool The infobox section of Wikipedia presents a summary of the key features and is used to improve navigation to other interrelated articles. The DBpedia extraction tool sometimes extracts the information incorrectly from the infobox that leads to the outliers.

```
dbr:Oldsmobile_88            dbo:wheelbase    1969^^xsd:integer.
dbr:Media,_Pennsylvania      dbr:postalCode   190631906519091^^xsd:integer.
```

Listing 1.4: Incorrect values extracted by DBpedia extraction tool

1. **Extracting value present at first place from Wikipedia Infobox**
   While going through outliers, we observed that DBpedia extraction tool extracts the numeric value present at the starting position place in the infobox. The Listing 1.4 shows that DBpedia extracted the information from Wikipedia about the car, namely Oldsmobile 88[7]. The actual value of wheelbase of Oldsmobile 88 is 124 inches whereas DBpedia extracted year 1969 present at the starting position.
2. **Concatenation of numbers**
   Sometimes DBpedia extraction tool removes the hyphens, comma or other separators between the values and merges them into one number. e.g. The

[7] https://en.wikipedia.org/wiki/Oldsmobile_88

maximum length of the postal code is ten digits worldwide [8]. However, in the Listing 1.4, it is shown that Media Pennsylvania has postal code '190631906519091' .

3. **Problems Interpreting and Converting Units**
There are some properties in DBpedia dataset which use different units of measurement. While converting these values, it can do some incorrect calculation. For example, it has extracted hyphen as a negative sign appended to the value. e.g. If a TV show duration is 11-12 and 24-26 minutes. The DBpedia extraction tool extracts values for runtime property in minutes i.e., -12 minutes from the infobox of Arthur TV series and the resulting runtime is (-12*60= -720 seconds).
4. **Infobox properties without starting year**
Some events on Wikipedia have no starting date, and the value for those events is empty and is written like "- 1998" DBpedia extracts this value as a negative year.
5. **Removing characters from alphanumeric value** [9]
In some observations DBpedia extraction tool removes the letters from alphanumeric values e.g. idoxifene with property unii has value 456UXE9867, and in DBpedia, it is stored as 456 only. Also IUPAC name is alphanumeric, but DBpedia extracted it as -88.

6 Conclusions and future work

In this paper, we have presented CONOD a simple, generic and scalable method to discover numeric outliers from KGs. CONOD is the first open source and distributed outlier detection algorithm for knowledge graphs. It makes use of Local similarity hashing to achieve faster cohorting. We have analyzed the results of CONOD for scalability and it has been found scalable up to a dataset of 128GB size on the available cluster. We have analyzed the resulting outliers both manually and graphically. The analysis shows the validity of the resulted outliers and also helped us in pointing out a few limitations of DBpedia extraction framework. CONOD can be extended by addition of more features prior to cohorting to obtain more meaningful cohorts. We can also choose from a range of different vectorization methods. An important branch to explore would to include other data types at literal positions for outlier detection. We can also explore the fact that the outliers can be present at the subject and predicate position of the triples by making use of semantic relationships to detect these outlier values.

Acknowledgment

This work was partly supported by the EU Horizon2020 projects WDAqua (GA no. 642795), Boost4.0 (GA no. 780732) and BigDataOcean (GA no. 732310).

[8] https://en.wikipedia.org/wiki/Postal_code
[9] https://en.wikipedia.org/wiki/Idoxifene

References

1. Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):15, 2009.
2. Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.
3. Jeremy Debattista, Christoph Lange, and Sören Auer. A preliminary investigation towards improving linked data quality using distance-based outlier detection. In *Joint International Semantic Technology Conference*, pages 116–124. Springer, 2016.
4. Daniel Fleischhacker, Heiko Paulheim, Volha Bryl, Johanna Völker, and Christian Bizer. Detecting errors in numerical linked data using cross-checked outlier detection. In *International Semantic Web Conference*, pages 357–372. Springer, 2014.
5. Frank E Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.
6. Tomáš Kliegr. Linked hypernyms: Enriching dbpedia with targeted hypernym discovery. *Web Semantics: Science, Services and Agents on the World Wide Web*, 31:59–69, 2015.
7. Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick Van Kleef, Sören Auer, et al. Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web*, 6(2):167–195, 2015.
8. Jens Lehmann, Gezim Sejdiu, Lorenz Bühmann, Patrick Westphal, Claus Stadler, Ivan Ermilov, Simon Bin, Nilesh Chakraborty, Muhammad Saleem, Axel-Cyrille Ngonga Ngomo, et al. Distributed semantic analytics using the sansa stack. In *International Semantic Web Conference*, pages 147–155. Springer, 2017.
9. Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764–766, 2013.
10. Robert McGill, John W Tukey, and Wayne A Larsen. Variations of box plots. *The American Statistician*, 32(1):12–16, 1978.
11. André Melo, Martin Theobald, and Johanna Völker. Correlation-based refinement of rules with numerical attributes. In *FLAIRS Conference*, 2014.
12. Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.
13. Heiko Paulheim. Identifying wrong links between datasets by multi-dimensional outlier detection. In *WoDOOM*, pages 27–38, 2014.
14. Dominik Wienand and Heiko Paulheim. Detecting incorrect numerical data in dbpedia. In *European Semantic Web Conference*, pages 504–518. Springer, 2014.
15. Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, 2012.