A Semantic Measure for Outlier Detection in Knowledge Graph

Bara Diop, Cheikh Talibouya Diop, Lamine Diop

A Semantic Measure for Outlier Detection in Knowledge Graph

Bara DIOP[1], Cheikh Talibouya DIOP[1], Lamine DIOP[2]

[1]University Gaston Berger of Saint-Louis, Senegal
[2]University of Tours, France

*E-mail : {diop.bara,cheikh-talibouya.diop}@ugb.edu.sn, lamine.diop@univ-tours.fr

Abstract

Nowadays, there is a growing interest in data mining and information retrieval applications from Knowledge Graphs (KG). However, the latters (KG) suffer from several problems related to data quality such as completeness, correctness and different kinds of errors. In DBpedia, there are several issues related to data quality. Among them, we focus on the following : several entities are in classes to which they do not belong. For instance, a query to get all the entities of the class $Person$ also returns groups, whereas these should be in the class $Group$. We call such entities "outliers". Discovery of such outliers is very important for class learning and understanding. In this paper, we propose a new outlier detection method that finds these entities. We define a semantic measure that favors the real entities of the class (inliers) with positive values while penalizing outliers with negative values and improve it with the discovery of frequent and rare itemsets. Our measure outperforms FPOF (frequent pattern outlier factor) ones. Experiments show the efficiency of our approach.

Keywords

Knowledge graph; Pattern Mining; Itemset; Outlier Detection

I INTRODUCTION

Most Semantic Knowledge Graphs (KG) are defined as hierarchy of classes (types), where entities may belong to several classes. For instance, (in figure 1, we can see that entity Bob_Marley belongs to classes $Artist$, $Person$ and $Agent$).... Hence, type prediction problem can be reformulated as hierarchical multi-label classification one [14]. In this context, one solution for entity type prediction is local classifier per node approach which consists in training separately each node of the class hierarchy, and determine the overall type. However, it has been observed that type information and more generally, KG suffer from several issues related to data quality such as completeness, correctness and different kinds of errors. Among them, we focus on the following : several entities are in classes to which they do not belong. For instance, a query to get all the entities of the class $Person$ class also returns groups, whereas these should be in the class $Group$. We call such entities "outliers". Discovery of such outliers is very important for correct class learning and understanding. Several outlier factors based on frequent pattern discovery were proposed [5, 9]. The idea behind is the following : transactions that contain more frequent patterns will have a big value of FPOF measure [5] and are unlikely to be outliers. In contrast, transactions with small FPOF values are likely to be outliers. Let's consider the following three entities of class $Artist$: https://dbpedia.org/page/Masaba_Gupta,

https://dbpedia.org/page/Bertram_Goodman and https://dbpedia.org/page/Mystik. Masaba Gupta and Bertram Goodman are real artists while Mystik is an outlier because it's a song. However, as we will see in example 4 , FPOF measure gives a value higher for Mystik than for Masaba Gupta and Bertram Goodman. FPOF does not take into account semantics aspects of the entities, while these are very important in KG and specifically in DBpedia. We propose a semantic measure that corrects such contradictions.

The main contributions of our paper are as follows :

- We define a semantic measure that favors the real entities of the class (inliers) with positive values while penalizing outliers with negative values and improve it with the discovery of frequent and rare itemsets.
- We propose a generic algorithm for outlier detection in knowledge graph, based on the semantic measure. This algorithm can be used for frequent or rare patterns.
- We present a set of experiments on dbpedia classes showing that our method outperforms FPOF like ones.

The rest of the paper is organized as follows. Section II discusses about related work. In section III, we propose the basic definitions and problem reformulation. Section IV presents the definition of FPOF in our context and shows its drawbacks by an example. In section V, we propose our measure and present a generic algorithm for outlier detection in knowledge graph while presenting a theoretical analysis. Section VI presents the results of our experiments and Section VII concludes the paper.

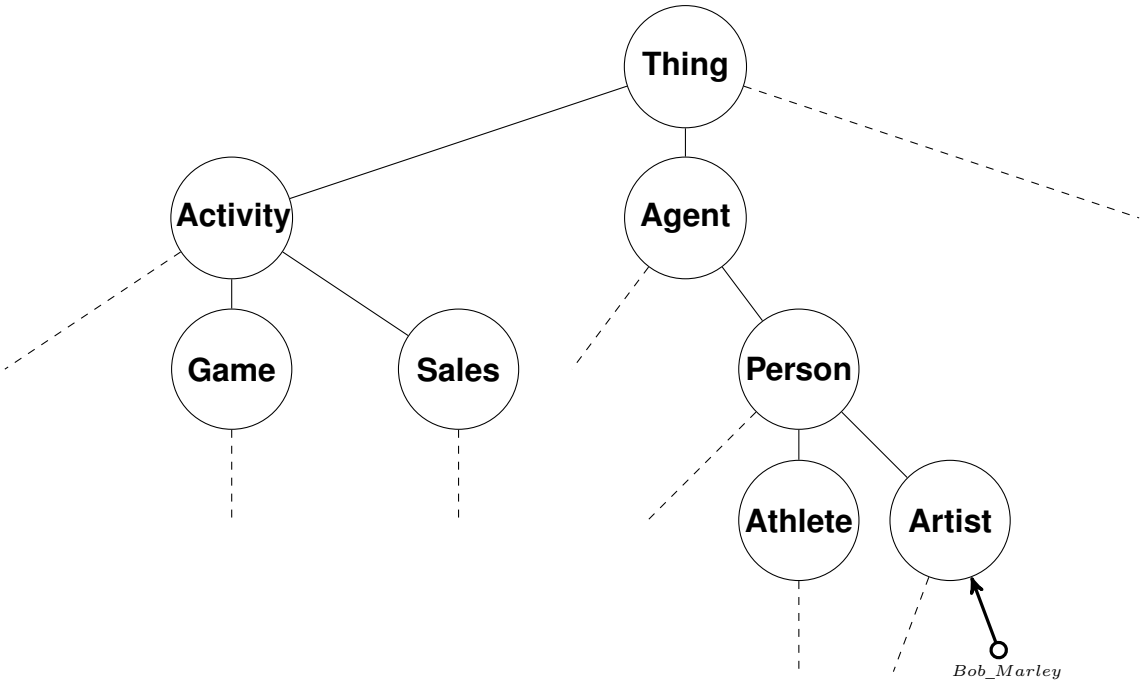


Figure 1: A sub-part of the DBpedia hierarchy

II RELATED WORK

Outlier detection has been the topic of a number of surveys and reviews. In these surveys, different methodology classifications are presented. More cited categories are the following : Nearest Neighbour Based Outlier Detection techniques, Distance-Based Techniques, Density-Based Techniques, Cluster Based Techniques, Statistical Approach Based Techniques. Distance-Based

[1] and Nearest Neighbour Based Techniques [4] rely on the notion of distance. A distance-based outlier in a dataset $\mathcal{D}$ is a data object with a given percentage of the objects in $\mathcal{D}$ having a distance of more than d_{min} away from it [9]. Nearest Neighbour based outlier detection techniques require a distance or similarity mesasure between two data points. If a point x has a short distance to its k neighbors, it is considered as normal otherwise it is considered as outlier. Density-based techniques measure density of a point x within a small region by counting number of points within a neighborhood region. Breunig et al. [2] introduced the concept of Local Outlier Factor (LOF), a score which is assigned to every point based on its local density. All data points are sorted in decreasing order of LOF value. Points with high scores are detected as outliers. In Cluster Based Outlier detection techniques, a cluster represents a collection of data objects similar to one another within the same cluster and dissimilar to the objects in other clusters. Inliers correspond to data in a cluster while outliers do not belong to any cluster. Statistical approach based technique assumes a distribution or probability model for the given data and then identifies outliers with respect to the model using a discordancy test [6]. Many of these techniques suffer from high dimensional space curse and high computational cost [13]. More recently, a new trend has appeared, using frequent pattern technique. Several measures were consequently proposed. FPOF (Frequent Pattern Outlier Factor) has been presented by [5]. The idea behind this measure is the following : transactions that contain more frequent patterns will have a big value of FPOF measure and are unlikely to be outliers. In contrast, transactions with small FPOF values are likely to be outliers. [9] proposed another measure WCFPOF (Weighted Closed Frequent Pattern Outlier Factor) in order to overcome the drawbacks of FPOF. According to this measure, transactions that contain more closed frequent patterns are more likely to be inliers and those that contain less closed frequent patterns are likely to be outliers. Another approach for overcoming the drawbacks of FPOF measurement was formulated by [10]. According to this one, transactions that contain longer frequent pattern (i.e. longer superset) are more likely to be inliers because they contain more subset frequent patterns, while transactions that short frequent patterns are likely to be outliers. The observation is the same for these different measures : they do not take into account the semantics.

III PRELIMINARIES AND PROBLEM REFORMULATION

Knowledge graph. A knowledge graph is a RDF dataset described by a set of triples $(subject, predicate, object)$. It can be represented as a tuple $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ where $\mathcal{T}$ is called a Tbox formed by names and assertions about concepts (or classes) and roles (or predicates), and $\mathcal{A}$ is called an ABox formed by assertions about individuals called entities and facts. In this paper, we focus on DBpedia[8] TBox and its entities. For example, $(dbo:Artist, rdfs:subClassOf, dbo:Person)$ is an assertion in DBpedia TBox which means that the concept $Artist$ is a subclass of the concept $Person$ (or $Person$ is a superclass of $Artist$). In other words, all artists are also persons. For example, $(Bob\_Marley, rdf:type, dbo:Artist)$ means that Bob_Marley is an artist, then an entity of the class $Artist$. The triple ***(Bob_Marley, spouse, Rita_Marley)*** is an assertion in DBpedia ABox which means that Bob_Marley has spouse $Rita\_Marley$. Given a class $C \in \mathcal{T}$ and an entity $e \in \mathcal{A}$, $(e, rdf:type, C)$ implies that e is an instance of C. To better formalize and define all these notions and the others that we need to formulate our problem, we use the Description Logics (DL) [7] formal notations. In that case, the assertion $(dbo:Artist, rdfs:subClassOf, dbo:Person)$ is denoted by $dbo:Artist \sqsubseteq dbo:Person$ which means that dbo:$Artist$ is subsumed by dbo:$Person$. dbo:$Artist(Bop\_Marley)$ denotes the fact that Bop_Marley is an instance (or entity) of the class $Artist$. The relation $spouse(Bob\_Marley, Rita\_Marley)$ materializes that the predicate

spouse links the subject Bob_Marley to the object $Rita\_Marley$. From these examples, we distinguish two types of predicates: outgoing and incoming. If we have the triple (X, P, Y), then we say that P is an outgoing predicate for X and an incoming predicate for Y whatever the type of X and Y. Then, there is a case where an outgoing predicate P is specific to subjects that instantiate a class C. In that case, we say that C is the domain of P, denoted by $\exists P.\top \sqsubseteq C$. Another case is when an incoming predicate is specific to objects of a class C. Here, we say that C is the range of the predicate P, denoted by $\top \sqsubseteq \forall P.C$. The disjointness of two classes C_1 and C_2 is denoted by $C_1 \sqsubseteq \neg C_2$. For example, according to the ontology of DBpedia, it should not have entities belonging to both the class $Person$ and the class $Organization$, then we have $Person \sqsubseteq \neg Organization$.

Transactional database. Now, we are going to show how to represent a transaction from the predicates that describe an entity. Since we are interested in predicates having domains, we consider the transactional database defined on the set of items $\mathcal{I} = \{P : (\exists C \in \mathcal{T})(\exists P.\top \sqsubseteq C)\}$. An itemset (or pattern), denoted by φ, is a none empty subset of $\mathcal{I}$. Formally, we have $\varphi \subseteq \mathcal{I}$. The set of all patterns that can be generated from $\mathcal{I}$ is called the pattern language $\mathcal{L} = 2^{\mathcal{I}} \setminus \emptyset$. In this paper, a transaction is a couple $(e, \mathcal{I}^e) \in \mathcal{A} \times \mathcal{L}$ where $\mathcal{I}^e$ is the set of all predicates describing the entity e that appear in $\mathcal{I}$. Formally, we have $\mathcal{I}^e = \{P \in \mathcal{I} : (\exists C \in \mathcal{T})(C(e))(\exists P.\top \sqsubseteq C)\}$. In the following, we simply denote such transaction as $\mathcal{I}^e$. So in our context, a transactional database $\mathcal{D}_C$ is a multi-set of transactions defined in $\mathcal{I}$ where all items (predicates) of $\mathcal{I}$ describes an entity of the class C. It means that we are making a restriction in class C in which we look for whether it contains outliers or not. For example, to find entities that are outliers in the class $Artist$, we only consider entities that instantiate $Artist$.

Example 1:
For instance, $\mathcal{D}_{Artist}$ in Table 1 is a toy dataset of 20 transactions from the class Artist of DBpedia[1](entities are prefixed by dbr(http://dbpedia.org/resource/) and predicates by dbo(http://dbpedia.org/ontology/)). For instance, to obtain the transaction $\mathcal{I}^{Bob\_Marley}$, we run the following SPARQL query on the Dbpedia endpoint (https://dbpedia.org/sparql/):

```
SELECT DISTINCT ?P WHERE {
   <http://dbpedia.org/resource/Bob_Marley> ?P ?object.
   ?P rdfs:domain ?C.
}
```

$\mathcal{D}_{Artist}$ is built from the set of 38 items $\mathcal{I} = \{$*deathPlace, deathDate, birthPlace, birthDate, deathCause, partner, relative, child, parent, spouse, birthYear, deathYear, birthName, nationality, field, training, residence, restingPlacePosition, restingPlace, cinematography, director, producer, starring, writer, runtime, Work/runtime, artist, endingTheme, network, openingTheme, previousWork, starring, subsequentWork, completionDate, numberOfEpisodes, bandMember, formerBandMember, hometown*$\}$. Table 2 gives the domain and the range (if it exists) of the set of items (predicates) in $\mathcal{I}$.

The transaction $\mathcal{I}^{Bob_Marley} = \{$deathPlace, deathDate, birthPlace, birthDate, deathCause, partner, relative, child, parent, spouse, birthYear, deathYear$\}$ contains 12 items. so, there are $2^{12} - 1$ patterns that appear in $\mathcal{L}(\mathcal{D}_{Artist})$. For example, $\varphi = \{spouse, birthYear\}$ is one among them. This pattern belongs also to the transaction $\mathcal{I}^{Hank\_Williams}$.

[1]Access in 18/10/2021

Table 1: $\mathcal{D}_{Artist}$: an example of transactional database from the class Artist of DBpedia

entity	Itemset (set of predicates)
Bob_Marley	{deathPlace, deathDate, birthPlace, birthDate, deathCause, partner, relative, child, parent, spouse, birthYear, deathYear}
Omar_Kiam	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}
Bertram_Goodman	{field, training}
Giuliana_Camerino	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}
Twice_as_Nice	{cinematography, director, producer, starring, writer, runtime, Work/runtime}
Robin_Harris	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}
Brett_Newski	{birthDate, hometown}
LaWanda_Page	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}
Frank_Suero	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}
Josephus_Thimister	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}
Sid_James	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}
Masaba_Gupta	{residence, spouse}
Hank_Williams	{deathPlace, deathDate, birthPlace, birthDate, restingPlacePosition, deathCause, relative, restingPlace, spouse, birthName, birthYear, deathYear}
Children_of_Eve	{cinematography, director, producer, writer, runtime, Work/runtime}
Greg_Giraldo	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}
Mystik	{artist, previousWork, producer, writer, runtime, Work/runtime}
Jerry_Clower	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}
Mackenzie_Taylor	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}
The_Lead	{endingTheme, network, openingTheme, previousWork, starring, subsequentWork, completionDate, numberOfEpisodes, runtime, Work/runtime}
7icons	{bandMember, formerBandMember, hometown}

Table 2: Domain and range (optional) of the predicates of our dataset $\mathcal{D}_{Artist}$ (EduIns: EducationalInstitution, TShow: TelevisionShow, Bcaster: Broadcaster, nonNegInt: nonNegativeInteger)

P	dom(P)	range(P)
deathPlace	Person	Place
deathDate	Person	date
birthPlace	Person	Place
birthDate	Person	date
deathCause	Person	
partner	Person	Person
relative	Person	Person
child	Person	Person
parent	Person	Person
spouse	Person	Person

P	dom(P)	range(P)
birthYear	Person	gYear
deathYear	Person	gYear
birthName	Person	langString
nationality	Person	Country
field	Artist	
training	Artist	EducationalInstitution
residence	Person	Place
restingPlacePosition	Person	SpatialThing
restingPlace	Person	Place
cinematography	Film	Person

P	dom(P)	range(P)
director	Film	Person
producer	Work	Agent
starring	Work	Actor
writer	Work	Person
runtime	Work	double
Work/runtime	Work	minute
artist	MusicalWork	Agent
endingTheme	TShow	Work
network	Bcaster	Bcaster

P	dom(P)	range(P)
openingTheme	TelevisionShow	Work
previousWork	Work	Work
subsequentWork	Work	Work
completionDate	Work	date
numberOfEpisodes	TShow	nonNegInt
bandMember	Band	Person
formerBandMember	Band	Person
hometown	Agent	Settlement

We already remark that, semantically, some of these entities such as *Twice_as_Nice*, *Children_of_Eve*, *Mystik* and *The_Lead* are not artists and yet they are instantiated in the Artist class. We call them outliers while the other are well instantiated in Artist, they are inliers.

Definition 1: Outlier, Inlier
Given two disjoint classes C and C' ($C \sqsubseteq \neg C'$) and an entity e. If e is instantiated in C but really defined as an entity of C', then e is an outlier. But if really e is an instance of C, then it is called an inlier.

Definition 2: Frequency of a pattern
Given a transactional database $\mathcal{D}_C = \{(e_1, \mathcal{I}^{e_1}), \cdots, (e_n, \mathcal{I}^{e_n})\}$ of a class C and a pattern $\varphi \in \mathcal{L}(\mathcal{D}_C)$. The frequency of φ is the number of transactions of $\mathcal{D}$ containing φ. Formally,

$$freq(\varphi, \mathcal{D}_C) = |\{(e_i, \mathcal{I}^{e_i}) \in \mathcal{D}_C : \varphi \subseteq \mathcal{I}^{e_i}\}|.$$

Example 2:
The frequency of $\varphi = \{spouse, birthYear\}$ in $\mathcal{D}_{Artist}$ is 2 because only $\mathcal{I}^{Bob\_Marley}$ and $\mathcal{I}^{Hank\_Williams}$ contain φ. The frequency of $\varphi_1 = \{spouse\}$ is $freq(\varphi_1, \mathcal{D}_{Artist}) = |\{Masaba\_Gupta, Bob\_Marley, Hank\_Williams\}| = 3$.

According the frequency, one can judge a pattern as frequent or rare (not frequent) in a dataset.

Definition 3: Frequent and Rare pattern
Let's consider a dataset $\mathcal{D}_C$ of a class C, a minimum support threshold $\alpha \in [0, 1]$ and φ a pattern of $\mathcal{L}(\mathcal{D}_C)$. We say that :

- φ is frequent in $\mathcal{D}_C$ if and only if $freq(\varphi, \mathcal{D}_C) \geq \alpha \times |\mathcal{D}_C|$.
- φ is rare in $\mathcal{D}_C$ if and only if $0 < freq(\varphi, \mathcal{D}_C) < \alpha \times |\mathcal{D}_C|$.

Example 3:
If we consider a minimum threshold $\alpha = 4/20$, then $\varphi = \{spouse, birthYear\}$ is a rare pattern in $\mathcal{D}_{Artist}$ because $freq(\varphi, \mathcal{D}_{Artist}) = \frac{2}{20} < \frac{4}{20}$ but the pattern $\varphi' = \{birthDate, birthName\}$ is frequent in $\mathcal{D}_{Artist}$ because $freq(\varphi, \mathcal{D}_{Artist}) = \frac{11}{20} > \frac{4}{20}$.

In this paper, we focus on two interestingness measures *frequency* and *rare*. So, given an interestingness measure m it's possible to extract a set of interesting patterns. We denote by $q(\cdot)$, such that $q(\cdot) \in \{true, false\}$, the constraint that a pattern of $\mathcal{D}_C$ should respect according the interestingness measure m. Therefore, on the one hand, the set of all frequent patterns given a minimum threshold α can be formulated by $\mathcal{P}set(\mathcal{D}_C, freq_\alpha) = \{\varphi \in \mathcal{L}(\mathcal{D}_C) : q(freq(\varphi, \mathcal{D}_C) \geq \alpha) = true\}$. On the other hand, the set of all rare patterns given a minimum threshold α can be formulated by $\mathcal{P}set(\mathcal{D}_C, rare_\alpha) = \{\varphi \in \mathcal{L}(\mathcal{D}_C) : q(0 < freq(\varphi, \mathcal{D}_C) < \alpha) = true\}$.

The problem that we want to solve in this paper can be formulated as follows:

Given a class C of knowledge graph $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, let $\mathcal{D}_C$ be its transactional database, and $\mathcal{P}set(\mathcal{D}_C, m_\alpha)$ a set of interesting patterns built with an interestingness measure m,

Q1: which metric can be used to favor the inliers and penalize the outliers?

Q2: how to benefit from the semantic of the TBox $\mathcal{T}$ in order to improve the metrics?

IV FPOF : FREQUENT PATTERN BASED OUTLIER FACTOR

As it was introduced in Section II, FPOF is based on frequent patterns. The main idea behind this approach for transactional databases is that, inliers will contain most of the frequent patterns while most of the patterns that outliers contain will have low frequency. So, they formulate the following metric.

Definition 4:
Given a transactional database $\mathcal{D}_C = \{(e_1, \mathcal{I}^{e_1}), ..., (e_n, \mathcal{I}^{e_n})\}$ and a minimal frequent $\alpha > 0$, the frequent pattern outlier factor of the transaction $(e, \mathcal{I}^e)$ denoted by $fpof(e, \mathcal{D}_C)$ is defined as follows:

$$fpof(e, \mathcal{D}_C) = \frac{1}{|\mathcal{P}set(\mathcal{D}_C, freq_\alpha)|} \times \sum_{\varphi \in \mathcal{P}set(\mathcal{D}_C, freq_\alpha) \wedge \varphi \subseteq \mathcal{I}^e} \frac{freq(\varphi, \mathcal{D}_C)}{|\mathcal{D}_C|}$$

Example 4:
Let's consider database $\mathcal{D}_{Artist}$ shown in Table 1. Each entity being described by a set of properties. For example, the entity Sid_James is described by the properties birthDate, birthName, deathDate, birthPlace and nationality. Considering the minsup 0.2 (4/20), the fpof values are calculated in Table 3 (column 3).

Based on the frequency of the itemsets, the FPOF measure gives a higher score to the entity $Mystik$ (which is an outlier) than to entities $Masaba\_Gupta$ and $Bertram\_Goodman$ (which are inliers). The FPOF measure is based only on the frequency of items that describe transactions while ignoring the semantics. In knowledge graphs, the frequency of items only does not allow to find the outliers of a set because semantics are decisive for entity definition. For example, $birthName$ is a predicate that defines a person which is less frequent than the predicate $abstract$ which is generally used by the entities of all classes.

V ONTOLOGY AND PATTERN-BASED OUTLIER DETECTION IN KNOWLEDGE GRAPH

In this section, we show how to take advantage of semantic relationships that arise in ontology in order to improve the mined pattern according to a measure of interest. But, let's start by introducing the idea behind the rare patterns for detecting outliers.

In a similar way to the FPOF metric, the rare patterns can be used to detect outliers from transactional database. A naive method is to sum, for any transaction, the frequency of the rare patterns it contains. In that case, the outliers will have high scores while the inliers will have low scores. But, the main problem with this basis metric follows from the fact that many inliers will have null scores. This is why in this paper, we are going to introduce semantics on metrics and particularly on items forming patterns which are nothing other than properties that appear in the ontology. It is important to note that all the properties that we use to make up the patterns have domains and some of them also have ranges. In the rest of this paper, the metrics as well as the algorithm that we are going to introduce are valid for both rare and frequent patterns.

Table 3: $\mathcal{D}_{Artist}$: an example of transactional database of the class Artist of DBpedia with fpof values

Artist	Properties	FPOF
Sid_James	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}	0,5129
Mackenzie_Taylor	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}	0,5129
Giuliana_Camerino	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}	0,5129
Jerry_Clower	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}	0,5129
Robin_Harris	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}	0,5129
Greg_Giraldo	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}	0,5129
LaWanda_Page	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}	0,5129
Frank_Suero	birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}	0,5129
Josephus_Thimister	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}	0,5129
Omar_Kiam	{birthDate, birthName, birthPlace, deathDate, deathPlace, nationality}	0,5129
Hank_Williams	{deathPlace, deathDate, birthPlace, birthDate, restingPlacePosition, deathCause, relative, restingPlace, spouse, birthName, birthYear, deathYear}	0,2704
Bob_Marley	{deathPlace, deathDate, birthPlace, birthDate, deathCause, partner, relative, child, parent, spouse, birthYear, deathYear}	0,1371
Brett_Newski	{birthDate, hometown}	0,0098
Children_of_Eve	{cinematography, director, producer, writer, runtime, Work/runtime}	0,0090
Mystik	{artist, previousWork, producer, writer, runtime, Work/runtime}	0,0091
Twice_as_Nice	{cinematography, director, producer, starring, writer, runtime, Work/runtime}	0,0091
The_Lead	{endingTheme, network, openingTheme, previousWork, starring, subsequentWork, completionDate, numberOfEpisodes, runtime, Work/runtime}	0,0091
Masaba_Gupta	{residence, spouse}	0
Bertram_Goodman	{field, training}	0
7icons	{bandMember, formerBandMember, hometown}	0

Intuition behind this method. Our approach is based on the fact that a property P with domain C must necessarily be an outgoing property of an entity of the class C when it is used. It is very important to note that P can also be an outgoing property of any entity of a subclass of C ($C' \sqsubseteq C$) or of a super-class of C located at a certain level $k \geq 0$. We say that a super-class C' of the class C is at level k with refer to C if there are $k-1$ classes $\{C_1, ..., C_{k-1}\}$, with $C_i \neq C_j$ for all $i \neq j$, in $\mathcal{T}$ such that $C \sqsubseteq C_1 \sqsubseteq ... \sqsubseteq C_{k-1} \sqsubseteq C'$. Indeed, the fact of favoring all the super-classes of C will distort the calculations because there will be outliers who will benefit from it. For example, *Artist* and *Athlete* are two classes in the same level that have super-classes Person (of level $k = 1$), Agent (of level $k = 2$) and Thing (of level $k = 3$). When searching for outliers in *Artist*, it is not interesting to consider properties that have Thing as its domain, because the latter is the parent class of all classes in DBpedia. However, properties that have *Agent* as their domain are only interesting if the outliers are in a twin class C' (*Activity* for instance) or one of sub-classes of C' (*Game*, *Sales*, ...). The same is true for properties that have Person as their domain, if some outliers really belong in class *Organization*, then they are useful for detecting outliers. If, on the other hand, the real class of the outliers is *Athlete*, then the properties having Person as their domain are not interesting to detect the outliers. So, the properties which can allow us to verify if an entity is an outlier or not, are not obvious. It's why retrieving these properties is a real problem of efficiency for our method because we do not know apriori at what level $k \geq 0$ we must stop to detect outliers since

we ignore their real classes. To solve this problem, we propose to vary the value of k to find the most relevant properties to detect outliers. In practice, the value of k is small. Therefore, we denote by $\mathcal{SC}\ell^{\leq k}(C) = \{C' \in \mathcal{T} : (\exists i \leq k)(C_1, ..., C_i \in \mathcal{T}^i)(C \sqsubseteq C_1 \sqsubseteq ... \sqsubseteq C_i)\}$ the set of super-classes of C which are located at a level less than or equal to k with respect to C. So, $\mathcal{SC}\ell^{\leq 0}(C) = \{C\}$ and $\mathcal{SC}\ell^{\leq \infty}(C) = \{C' \in \mathcal{T} : C' \sqsubseteq C\}$.

Let's denote by $\mathcal{C}^{\leq \star}(C)$ the set of concepts containing C, its super-classes at a level at most equal to k and its sub-classes: $\mathcal{C}^{\leq \star}(C) = \{C\} \cup \{C' \in \mathcal{T} : C \sqsubseteq C'\} \cup \{C' : C' \in \mathcal{SC}\ell^{\leq \star}(C)\}$. The semantic judgment we have on a property φ that appears in a pattern φ can be formulated as follows:

- (a) If a property P is interesting enough to well describe an inlier e of a class C in a certain level k then it must necessarily have a domain which is part of $\mathcal{C}^{\leq k}(C)$. The set of predicates in the pattern φ that meet this intuition is defined by $\{P \in \varphi : (\exists C' \in \mathcal{C}^{\leq k}(C))(\exists P.\top \sqsubseteq C')\}$.
- (b) An entity found in class C which has an incoming property P whose domain does not belong to $\mathcal{C}^{\leq \infty}(C)$ is likely to be poorly described, and looks like an outlier if most of the properties it contains have this tendency. The set of predicates in the pattern φ that meet this intuition is defined by $\{P \in \varphi : (\nexists C' \in \mathcal{C}^{\leq k}(C))(\exists P.\top \sqsubseteq C')\}$.
- (c) If a property P that matches the intuition in (a) additionally has a range that is in $\mathcal{C}^{\leq k}(C)$, then it is determinant for class C. The set of predicates in the pattern φ that meet this intuition is defined by $\{P \in \varphi : (\exists C' \in \mathcal{C}^{\leq k}(C))(\exists P.\top \sqsubseteq C')(\exists C'' \in \mathcal{C}^{\leq k}(C))(\top \sqsubseteq P.C'')\}$.
- (d) If a property P that matches the intuition in (b) additionally has a range that is part of $\mathcal{C}^{\leq k}(C)$, then the use of P is ambiguous. The set of predicates in the pattern φ that meet this intuition is defined by $\{P \in \varphi : (\nexists C' \in \mathcal{C}^{\leq k}(C))(\exists P.\top \sqsubseteq C')(\exists C'' \in \mathcal{C}^{\leq k}(C))(\top \sqsubseteq \forall P.C'')\}$.

Based on these intuitions, we can now define our algorithm which combines the mined pattern and the semantic relationships from ontology to detect outliers.

A generic algorithm for outlier detection in knowledge graph. The algorithm we propose here is based on the four intuitions presented in the previous section. To be in agreement with these intuitions, we say that a predicate which verifies intuition (a) brings a bonus of 1 to the score of the entity that it describes while that which verifies (b) brings a penalty of 1 to the entity's score. On the other hand, a predicate which verifies intuition (c) brings a bonus of 2 (bonus of 1 for the domain and of 1 for the range) while a predicate which verifies intuition (d) brings a penalty of 2 for the entity it describes (penalty of 1 for the domain and 1 for the range). Thereby, given a pattern we introduce the notion of reliability which is a metric to compute the total score a pattern brings to an entity with respect to a given class.

Definition 5: Reliability
Let e be an entity of a class C described by a certain predicates that belong to $\mathcal{I}$, a pattern $\varphi \subseteq \mathcal{I}$ and k a positive integer. The reliability of the pattern φ to the entity e with respect to the class

C is defined as follows:

$$reliability(\varphi, e, C, k) = \begin{cases} 0 \ \ if \ \ \varphi \not\subseteq \mathcal{I}^e \\ \\ |\{P \in \varphi : (\exists C' \in \mathcal{C}^{\leq k}(C))(\exists P.\top \sqsubseteq C')\}| \\ +|\{P \in \varphi : (\exists C' \in \mathcal{C}^{\leq k}(C))(\exists P.\top \sqsubseteq C')(\top \sqsubseteq P.C')\}| \\ -|\{P \in \varphi : (\nexists C' \in \mathcal{C}^{\leq \infty}(C))(\exists P.\top \sqsubseteq C')\}| \\ -|\{P \in \varphi : (\nexists C' \in \mathcal{C}^{\leq \infty}(C))(\exists P.\top \sqsubseteq C') \\ \ (\exists C'' \in \mathcal{C}^{\leq k}(C)(C))(\top \sqsubseteq \forall P.C'')\}| \ \ otherwise \end{cases} \quad (1)$$

It's clear that the reliability of a pattern is the sum of the contribution of all the predicates it contains. Now, we can present the algorithm that explore this metric in order to capture the outliers which disturb the veracity of all the instances contained in a given class. Let's recall that the goal of this paper is not to find frequent or rare patterns, we just reuse the existing methods.

Example 5:
Let's consider this two patterns $\varphi_1 = \{birthDate, birthName\}$ and $\varphi_2 = \{bandMember, hometown\}$. We are now going to compute the reliability of the transactions $(Robin\_Harris, \mathcal{I}^{Robin\_Harris})$ and $(7icons, \mathcal{I}^{7icons})$ with refer to the class *Artist* and with different values of k. We have :
$reliability(\varphi_1, Robin\_Harris, Artist, 0) = 0$ because none of the properties in φ_1 has a domain that belongs in $\mathcal{C}^{\leq 0}(Artist) = \{Artist\}$ but in $\mathcal{C}^{\leq \infty}(Artist)$. These predicates are neutral since they have a domain that is a superclass of *Artist*. However, $reliability(\varphi_1, Robin\_Harris, Artist, 1) = (+1) + (+1) = +2$ because birthDate and birthName have a domain that belongs in $\mathcal{C}^{\leq 1}(Artist) = \{Artist, Person\}$ then (+1). $reliability(\varphi_2, 7icons, Artist, 0) = -1$ because, in one hand, the domain of the property bandMember doesn't belong in $\mathcal{C}^{\leq \infty}(Artist)$ while its range doesn't belong in $\mathcal{C}^{\leq 0}(Artist)$. In other hand, the domain of the property hometown belongs in the set $\mathcal{C}^{\leq \infty}(Artist)$ but not in $\mathcal{C}^{\leq 0}(Artist)$. Let's now compute $reliability(\varphi_2, 7icons, Artist, 1)$. We know that the range of bandMember belongs in $\mathcal{C}^{\leq 1}(Artist)$ while its domain doesn't belong in $\mathcal{C}^{\leq \infty}(Artist)$. So, this property gives a score of ((-1)+(-1)) to the entity $7icons$. The property hometown gives the same score like in $k = 0$. So, we have $reliability(\varphi_2, 7icons, Artist, 1) = (-1) + (-1) = -2$.

Algorithm 1, named ONTOPOD, computes and returns a set of entities likely to be outliers. It takes as input the transactional database obtained from the entities of a class C of a given knowledge graph $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, an interestingness measure m_α, with α the minimum threshold, and a positive integer k for the maximum level of superclasses to be considered for C. In the end, it returns all the entities that have a negative score, and therefore likely to be outliers. Then, it computes the score of each entity e according its transaction $\mathcal{I}^e$ and the extracted patterns $\mathcal{P}set(\mathcal{D}_C, m_\alpha)$ if it is not empty (lines 2-8). Finally, all the entities having a negative score are retrieved and stored in the $setOut$ variable (line 5) which is finally returned at output (line 9). Interestingly, it's also possible to easily flip inliers (those with positive scores) in the $setIn$ variable (line 8). We maintain the idea that the method fails to judge entities that have a score equal to zero (0). We specifically use three variants of ONTOPOD. The first is ONTOPOD$-No$ which means that $\mathcal{P}set(\mathcal{D}_C, m_\alpha)$ is empty. The second is ONTOPOD $-$ $Freq$ that corresponds to the case where the interestingness measure is the frequency, $\mathcal{P}set(\mathcal{D}_C, freq_\alpha)$. The last is ONTOPOD $-$ $Rare$ which means that the rare patterns are used $\mathcal{P}set(\mathcal{D}_C, rare_\alpha)$.

Algorithm 1 ONTOPOD (Ontology and Pattern-based Outlier Detection

Input: A set of patterns $\mathcal{P}set(\mathcal{D}_C, m_\alpha)$ obtained from the transactional database $\mathcal{D}_C$ of a class C of a knowledge graph $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and an interestingness measure m_α with $\alpha \in [0, 1]$ and k a positive integer and a positive integer k

Output: A set of entities weigted by their scores $setOut$ (for outliers) and $setIn$ (for inliers) obtained from $\mathcal{D}_C$

$setOut \leftarrow \emptyset$ ▷ The set of outliers

$setIn \leftarrow \emptyset$ ▷ The set of inliers

for $(e, \mathcal{I}^e) \in \mathcal{D}_C$ **do**

  $score(e, C, k) \leftarrow reliability(\mathcal{I}^e, e, C, k) + \sum_{\varphi \in \mathcal{P}set(\mathcal{D}_C, m_\alpha) \setminus \{\mathcal{I}^e\} \wedge \varphi \subseteq \mathcal{I}^e} reliability(\varphi, e, C, k)$

  if $score(e, C) < 0$ **then**

    $setOut \leftarrow setOut \cup \{(e, score(e, C))\}$

  else

    $setIn \leftarrow setIn \cup \{(e, score(e, C))\}$

return $setOut$

Example 6:
Table 4 gives the scores of ONTOPOD applied in the dataset of Table 5 that corresponds to the Artist class by promoting the super-class at level 1 (Person) ($m_\alpha = no$ means that there is no interestingness measure that has been provided). As we can see it, our method manages to find outliers that the FPOF method failed to judge (Masaba_Gupta, Bertram_Goodman, 7icons). It is also important to note that without an interestingness measure, certain entities risk to have a score equal to zero (Brett_Newski), hence the interest in finding frequent or rare patterns for deciding. Moreover, with frequent patterns, ONTOPOD finds all the outliers while with rare patterns, it adds an inlier (Brett_Newski) to the outliers.

Theoretical analysis of Algorithm 1. In this section, we give the theoretical analysis of our algorithm ONTOPOD. As we said it earlier, we do not take into account the complexity of the pattern mining which depends on the used pattern mining algorithm. However, the complexity of our method also depends on the size of the set of mined patterns. Generally, to compute the score of an transaction $(e, \mathcal{I}^e)$, we first compute the reliability of the itemset $\mathcal{I}^e$ with refer to the corresponding entity e in $O(|\mathcal{I}|)$. Then, we compute the reliability of each pattern φ in $\mathcal{P}set(\mathcal{D}_C, m_\alpha)$ to a transaction $(e, \mathcal{I}^e)$ with a complexity in $O(|\mathcal{I}| \times |\mathcal{P}set(\mathcal{D}_C, m_\alpha)|)$. Thereby, the complexity of ONTOPOD is in $O(|\mathcal{I}| \times |\mathcal{D}_C|) + O(|\mathcal{I}| \times |\mathcal{D}_C| \times |\mathcal{P}set(\mathcal{D}_C, m_\alpha)|)$. So, the complexity of ONTOPOD is generally in $O(|\mathcal{I}| \times |\mathcal{D}_C| \times (1 + |\mathcal{P}set(\mathcal{D}_C, m_\alpha)|))$.

In the case of ONTOPOD–No, where the set of patterns $\mathcal{P}set(\mathcal{D}_C, m_\alpha)$ is empty, the complexity is only in $O(|\mathcal{I}| \times |\mathcal{D}_C|$.

VI EXPERIMENTATIONS

6.1 Datasets

For our experiments we use two DBpedia datasets: the online dataset and a benchemark dataset [15]. The benchemark dataset lacks information on the properties of the entities, which is why we use the online dataset.

Table 4: $\mathcal{D}_{Artist}$: an example of transactional database from the Artist class of DBpedia with fpof and ONTOPOD values

		ONTOPOD with $\mathcal{P}set(\mathcal{D}_{Artist}, m_\alpha)$ and $k = 1$		
Artist	FPOF	$m_\alpha = no$	$m_\alpha = freq_{0.2}$	$m_\alpha = rare_{0.2}$
Sid_James	0,5129	6	198	6
Mackenzie_Taylor	0,5129	6	198	6
Giuliana_Camerino	0,5129	6	198	6
Jerry_Clower	0,5129	6	198	6
Robin_Harris	0,5129	6	198	6
Greg_Giraldo	0,5129	6	198	6
LaWanda_Page	0,5129	6	198	6
Frank_Suero	0,5129	6	198	6
Josephus_Thimister	0,5129	6	198	6
Omar_Kiam	0,5129	6	198	6
Hank_Williams	0,2704	14	94	28606
Bob_Marley	0,1371	17	49	34801
Brett_Newski	0,0098	1	2	2
Children_of_Eve	0,0090	-9	-13	-293
Mystik	0,0091	-7	-11	-227
Twice_as_Nice	0,0091	-11	-15	-711
The_Lead	0,0091	-11	-15	-5639
Masaba_Gupta	0	3	3	9
Bertram_Goodman	0	2	2	6
7icons	0	-4	-4	-20

DBpedia Online: DBpedia Online is a set of human-understandable online RDF schema descriptions accessible to applications. The data is organized in a hierarchical structure. In this dataset we will extract, for a class C, its entities (name, properties, domain and range of properties), its super-classes and sub-classes. As already stated these classes contain outliers. we will use the benchmark dataset to test the performance of our methods.

Benchemark dataset: This benchemark dataset is a reference database containing 342,781 data instances that can be used for hierarchical classification tasks. It has 3 levels l1, l2, l3, with respectively 9, 70 and 219 classes. We will focus on some classes of level 2. This dataset contains the real classes of the entities but does not cover the data of the online database.

6.2 Protocol

6.2.1 Extracting information from DBpedia online

We have chosen to work with some DBpedia classes. These are the classes «Animal», «Artist», «Athlete», «Company», «EducationalInstitution», «Group», «Politician», and «NaturalPlace». For the online extraction, we first choose a class C of these classes and then use SPARQL queries to extract its entities, super-classes and sub-classes. The entities of the sub-classes of C are also part of the entities of C. The properties of each entity and the domains and ranges are also extracted. For instance, for the Artist class the following SPARQL queries were executed:

```
SELECT distinct ?e ?P ?domain ?range
WHERE {
```

```
   ?e a dbo:Artist.
   ?e ?P ?objet.
   ?P rdfs:domain ?domain.
   OPTIONAL { ?P rdfs:range ?range }
}
```

```
SELECT distinct ?C
WHERE {
   dbo:Artist rdfs:subClassOf* ?C.
}
```

```
SELECT distinct ?c
WHERE {
    ?c rdfs:subClassOf+ dbo:Artist.
}
```

After the extraction of the online information, for a class C, we keep only the resources having at least two properties which are in the gold standard since it is these resources whose real class is known.

In Table 5, we give the number of entities, the number of outliers and the number of inliers in each class. Each entity is described by a set of properties.

Table 5: Number of outlier and inlier entities in classes

$\mathcal{D}$	$\|Entities\|$	$\|Outliers\|$	$\|Inliers\|$
$Animal$	1991	20	1971
$Artist$	2484	96	2388
$Athlete$	3040	16	3024
$Company$	7848	1013	6835
$EducationalInstitution$	3477	15	3462
$Group$	1519	386	1133
$Politician$	6920	210	6710
$NaturalPlace$	8195	12	8183

6.2.2 *Extraction of frequent and rare patterns*

For the detection of frequent and rare patterns we used the $fpgrowth$ [3] algorithm to find all supports of the patterns in the database. Then we separated the database into frequent and rare patterns using the average support as a separation threshold.

6.2.3 *Measures to evaluate the performance of the algorithm*

For the evaluation of the methods, we used three measures: the inliers rate *IR*, the outliers rate *OR* and the harmonic mean of the *IR* and the *OR* called *F-score*. For each method m, we

calculate *IR(m)*, *OR(m)* and *F-score(m)*.

$$IR(m) = \frac{ni_m}{ni_T} \tag{2}$$

Where ni_m is the number of inliers found by the method m and ni_T is the total number of inliers in the database.

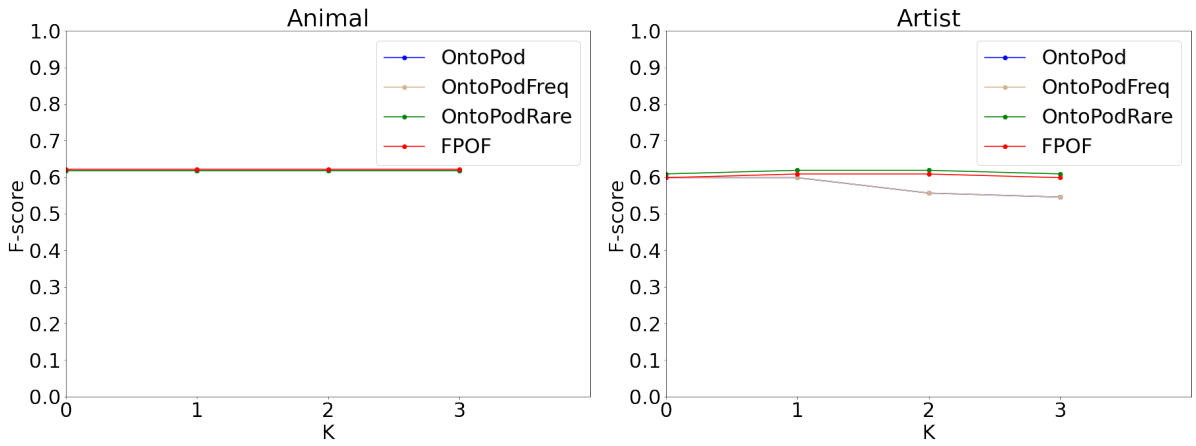
$$OR(m) = \frac{no_m}{no_T} \tag{3}$$

Where no_m is the number of outliers found by the method m and no_T is the total number of outliers in the database.

$$F - socre = 2 * \frac{IR * OR}{IR + OR} \tag{4}$$

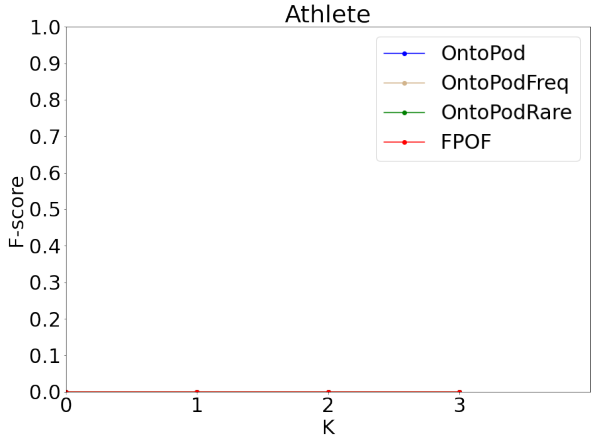
There are two sets of experiments that were conducted in each class to evaluate the performance the methods. The first one is the variation of the f-score of the methods according to the value of k and the second one is the outlier rate and the f-score of the methods for the best value of k.

6.3 Results

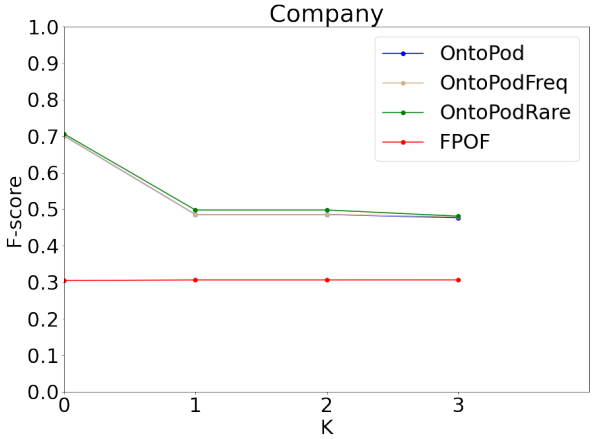


(a)

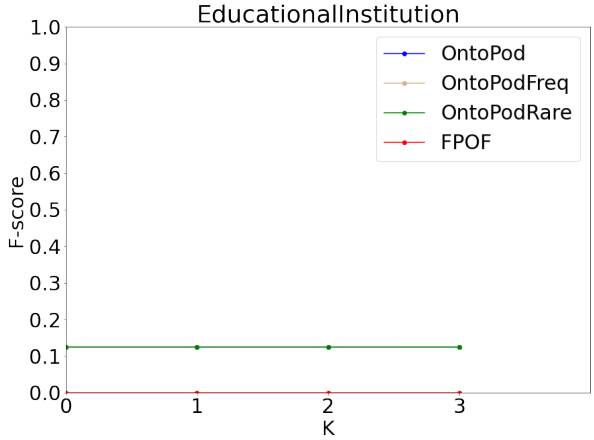
(b)



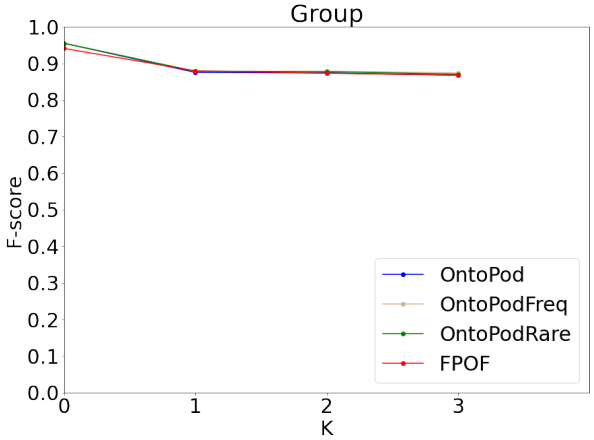
(c)



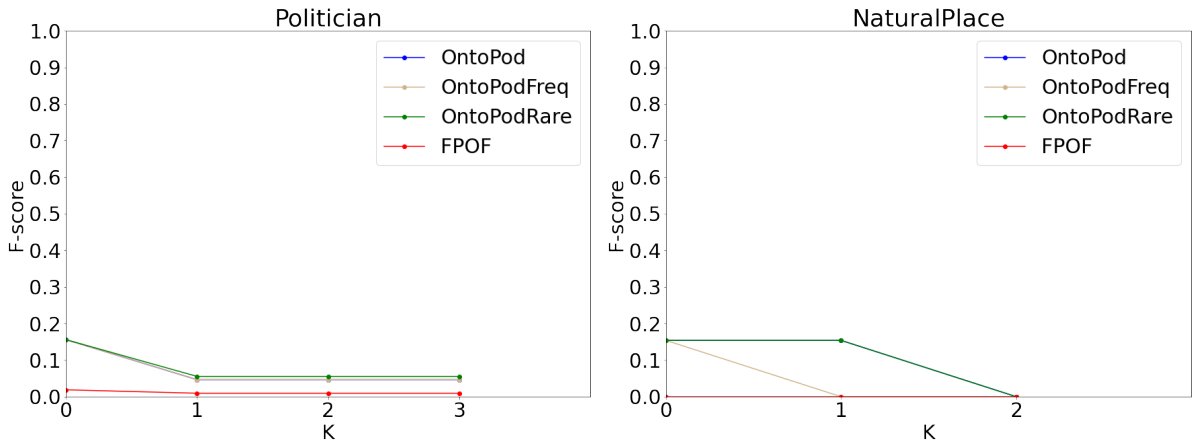
(d)



(e)

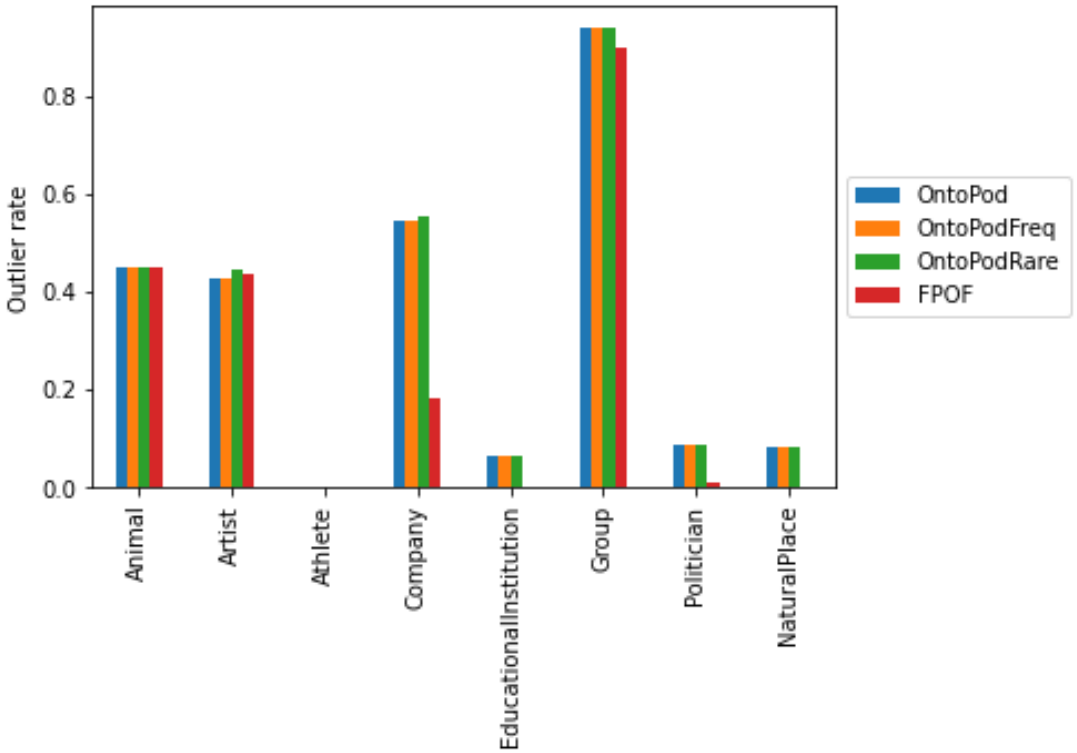


(f)

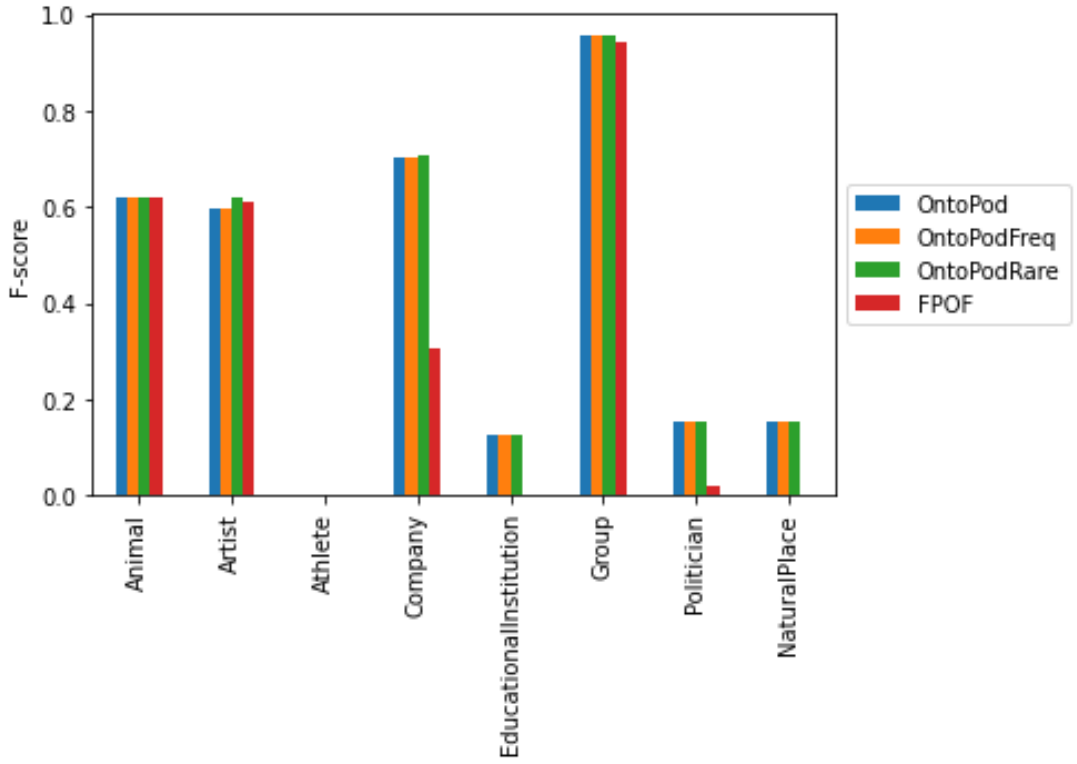


(g) (h)

Figure 2: F-score of methods for each class



(a)



(b)

Figure 3: Outlier rate and F-score per class

6.3.1 *Evolution of the f-scores of the methods according to the values of k*

Figure 2 shows the evolution of the f-scores of the methods according to the value of k. We notice that in general the best values of F-score correspond to $k = 0$. The figures show that our method outperforms the FPOF method. For some classes the fpof method can't even find outliers. Most outliers are entities of superclasses and their properties are very frequent.

6.3.2 *Performance of the methods on the different classes*

Figure 3 shows the performance of the methods on the different classes. Figure 3(b) is the result of figure 2 for the maximum f-score value (which corresponds in general to $k = 0$). Figure 3(a) shows the outlier rate for each method in each class. For the classes Company, EducationalInstitution, Politician and NaturalPlace our method largely outperforms the fpof method. The outliers have frequent properties that is why the fpof method fails to find them.

VII CONCLUSION

Knowledge graphs often suffer from several issues to data quality. However, good data quality has a significant impact on learning tasks. This paper proposes a semantic measure that favors real entities of the class and disfavors outliers, and an outlier detection algorithm based on this measure. Experiments have shown the effectiveness of our algorithm. In future work, we plan to test our algorithm on other datasets and do type classification on DBpedia by removing outliers.

REFERENCES

Publications

[1] E. M. Knorr and R. T. Ng. "Algorithms for Mining Distance-Based Outliers in Large Datasets". In: *VLDB*. 1998.

[2] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. "LOF: Identifying Density-Based Local Outliers". In: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*. SIGMOD '00. Dallas, Texas, USA: Association for Computing Machinery, 2000, pages 93–104. ISBN: 1581132174.

[3] J. Han, J. Pei, and Y. Yin. "Mining Frequent Patterns without Candidate Generation". In: *SIGMOD Rec.* 29.2 (May 2000), pages 1–12. ISSN: 0163-5808.

[4] S. Ramaswamy, R. Rastogi, and K. Shim. "Efficient Algorithms for Mining Outliers from Large Data Sets". In: *SIGMOD Rec.* 29.2 (May 2000), pages 427–438. ISSN: 0163-5808.

[5] Z. He, X. Xu, J. Z. Huang, and S. Deng. "FP-outlier: Frequent pattern based outlier detection". In: *Comput. Sci. Inf. Syst.* 2 (2005), pages 103–118.

[6] Z. abu bakar, R. Mohemad, A. Ahmad, and M. Mat Deris. "A Comparative Study for Outlier Detection Techniques in Data Mining". In: July 2006, pages 1–6.

[7] F. Baader, D. Calvanese, D. Mcguinness, D. Nardi, and P. Patel-Schneider. *The Description Logic Handbook: Theory, Implementation, and Applications*. Jan. 2007.

[8] C. Bizer, J. Lehmann, G. Kobilarov, S. Auer, C. Becker, R. Cyganiak, and S. Hellmann. "DBpedia - A crystallization point for the Web of Data". In: *Journal of Web Semantics* 7.3 (2009). The Web of Data, pages 154–165. ISSN: 1570-8268.

[9] J. Ren, Q. Wu, C. Hu, and K. Wang. "An Approach for Analyzing Infrequent Software Faults Based on Outlier Detection". In: *2009 International Conference on Artificial Intelligence and Computational Intelligence*. Volume 4. 2009, pages 302–306.

[10] W. Zhang, J. Wu, and J. Yu. "An Improved Method of Outlier Detection Based on Frequent Pattern". In: *2010 WASE International Conference on Information Engineering*. Volume 2. 2010, pages 3–6.

[11] G. Töpper, M. Knuth, and H. Sack. "DBpedia ontology enrichment for inconsistency detection". In: *ACM International Conference Proceeding Series* February 2016 (2012), pages 33–40.

[12] P. Vateekul. "Hierarchical Multi-Label Classification: Going Beyond Generalization Trees". In: (2012), page 209.

[13] A. M. Said, D. D. Dominic, and B. B. Samir. "Outlier Detection Scoring Measurements Based on Frequent Pattern Technique". In: *Research Journal of Applied Sciences, Engineering and Technology* 6 (2013), pages 1340–1347.

[14] A. Melo, H. Paulheim, and J. Völker. "Type Prediction in RDF Knowledge Bases Using Hierarchical Multilabel Classification". In: *Proceedings of the 6th International Conference on Web Intelligence, Mining and Semantics* (2016).

[15] D. Ofer. *DBPedia Classes : Hierarchical Taxonomy of Wikipedia article classes*. https://www.kaggle.com/danofer/dbpedia-classes. [Online; accessed 02-November-2021]. 2019.