Outlier Detection in Linked Open Data

Bachelor-Thesis von Dominik Wienand aus Darmstadt

Juni 2013

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Knowledge Engineering

Outlier Detection in Linked Open Data

Vorgelegte Bachelor-Thesis von Dominik Wienand aus Darmstadt

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: Dr. Heiko Paulheim

Tag der Einreichung: 3.6.2013

Table of Contents

1. Introduction

1.1 Abstract

Linked Open Data(LOD) makes a vast amount of information freely accessible. DBpedia is a central hub of this LOD cloud and scrapes data from structured Wikipedia elements to make them available to the Semantic Web. However, this scraping approach is prone to errors, which is why error detection methods and quality assessments are needed.
We study the application of general and numerical outlier detection methods to LOD. Methods investigated include Interquantile Range(IQR), Kernel Density Estimation(KDE) and various dispersion estimators. We analyze data types to identify structural errors and RDF types to segregate datasets for further analysis.
We manually evaluate the approaches on selected datasets as well as on random samples from DBpedia. The data collected during this evaluation is then used to assess some quality aspects of DBpedia.
Combinations of IQR and KDE with certain grouping strategies based on RDF types achieve about 87% precision on selected datasets as well as on random samples. Runtime-wise, however, we find vast differences that make KDE and more sophisticated clustering methods appear unfeasible for time-critical scenarios.
Regarding the data quality, we identify 11 different sources of errors in the DBpedia extraction framework along with various other inconsistencies. Between the *DBpedia-OWL* and *DBProp* namespaces, we find vast differences in quality.

1.2 Problem Statement

The goal of this Bachelor thesis is to find and evaluate methods of detecting and possibly correcting errors in Linked Open Data. Those methods should be unsupervised so they can be incorporated into the scraping and parsing methods already found on the Semantic Web.
In a second step we take a look at the outliers we discovered this way to identify common sources of errors on DBpedia.

This work focuses on DBpedia for multiple reasons:

1) As can be seen on Figure 1: LOD Cloud, DBpedia is a central hub of the Linked Web and thus one of the most important subjects to analyze
2) DBpedia, being sourced from Wikipedia, contains diverse data from all domains of life
3) The way DBpedia gathers its data from Wikipedia makes it susceptible to errors and thus DBpedia would specifically benefit from outlier detection

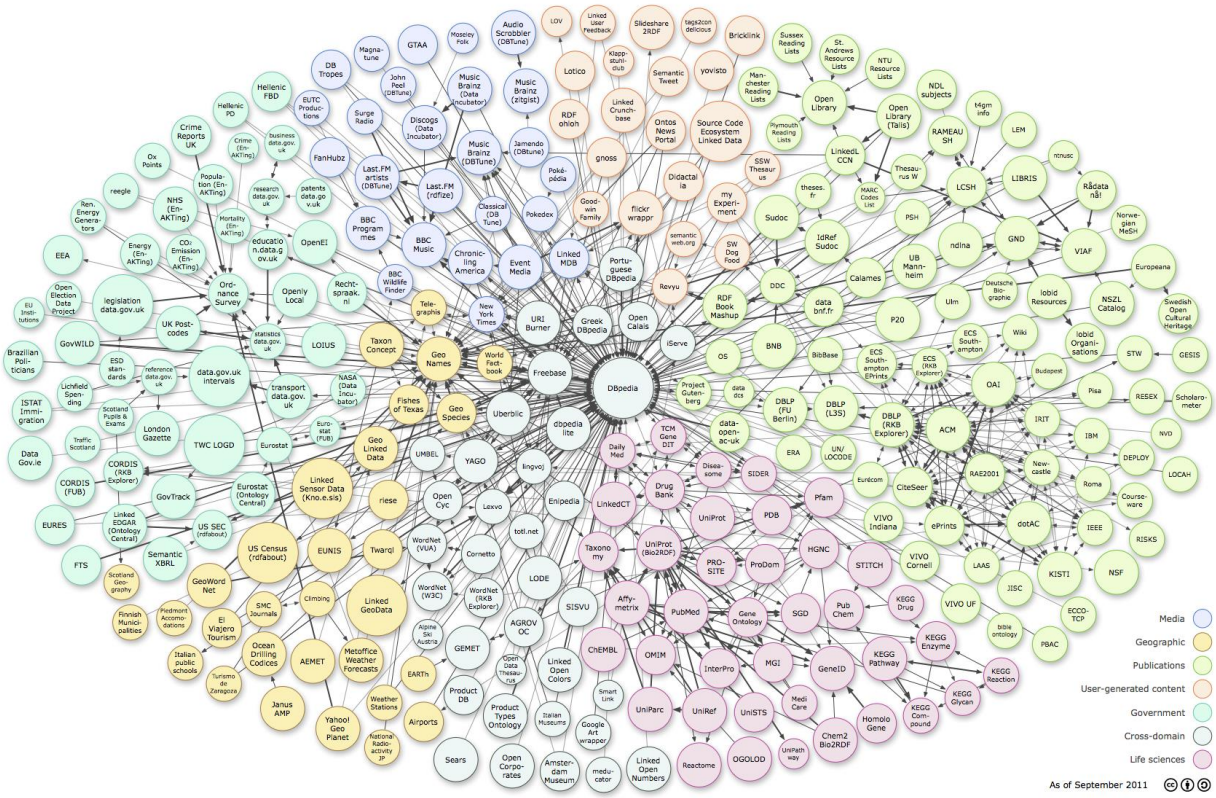

Figure 1: LOD Cloud (Cyganiak & Jentzsch, 2011)

The need for an unsupervised method combined with the extreme diversity of the data we are dealing with means our methods have to be robust but we cannot make many assumptions about the distribution of the data. Further complicating things is that we are frequently dealing with larger datasets (up to millions of data points), which makes efficiency important.

After evaluating various outlier detection approaches, we use the data we collected during the evaluation to find common sources of erroneous data points in DBpedia and to assess some more general aspects of DBpedia, such as how the *dbpedia-owl* namespace compares to the *dbpprop* namespace.

1.3 Overview

In section 2, we give background information about the Semantic Web and the statistics of outlier detection.
In section 3, we present the approaches that we considered to solve our problems.
In section 4, we evaluate those approaches.
In section 5, we use the data collected during the evaluation phase to perform quality analysis on DBpedia.
In section 6, we give a conclusion of our results and an outlook on possible further questions regarding the topics of this work.

2. Background

2.1 Semantic Web

The Semantic Web is an initiative by the World Wide Web Consortium (W3C) that promotes augmenting the World Wide Web with semantics that are machine-readable (W3C, 2013).
The World Wide Web is full of information. It has flourished through its decentralized and interlinked nature. Most of this information is meant to be specifically accessible to humans, unlike databases, which are primarily geared to machine processing. That means text is structured with headings, section, paragraphs etc. to help us quickly grasp its content and find the information we are looking for. This, however, does little to enable computers to access the information contained in the text.
If we want to know when and where a certain "Tom" was born, the following sentence provides all the information we need: "Tom was born on January 29, 1963 in the town of Exampleton." However, in order to process that information with a computer, it has to be available in a structured, standardized format, because there is a practically arbitrary number of different representations of that same information in natural language, all of which the computer would have to be able to parse automatically.

The basic structure of the WWW is provided by the HTTP protocol and the HTML markup language. HTML consists mostly of content between tags that specify *how* that content will be displayed but hardly any information about *what* that content represents.
This makes it hard for a machine to directly answer even simple questions such as "When was [some person] born?" "What is the capital of [some country/state]?" What search engines, given such a query, will usually do, is present us with a link to a corresponding homepage or Wikipedia article where we can then find the information we are looking for. However, if our query is somewhat more advanced, for example "find all known persons born before [some data] in [some capital]", this approach quickly comes to an end. Given a database with the all the relevant information, this query could easily be handled. And all the information is available on the WWW, because for every somewhat notable person there is probably a web page that will at least state the place and date of birth. But since there is no simple way to automatically aggregate and process that information, because it is completely unstructured, our query remains unanswered.

There are two basic approaches to overcome those limitations of the current web: Extracting information from the existing, unstructured data and creating new structured data in a machine friendly format. The former is commonly known as Data Mining, the latter is the approach of the Semantic Web.
The idea is to create a web, much like the WWW, of machine accessible information. The information is made machine accessible by augmenting it with semantics that a computer can make use of.

2.1.1 RDF

The "Resource Description Framework" (RDF) was created to model information graphs and forms one of the foundations of implementations of the Semantic Web idea (Klyne, Carroll, & McBride, 2004).
At its core, there are triples of Unique Resource Identifiers (URI) and literals. As the name implies, URIs are supposed to uniquely identify resources within their namespace. Beyond that, URIs are generally supposed to be dereferencable, that is, it should be possible to obtain the resource they

represent in some way. For practical purposes, that means retrieving them from the Internet via http; thus most URIs take the form of URLs such as `http://dbpedia.org/resource/Darmstadt`. Literals represent primitives such as `1` or `Germany`. It is possible to augment them with suffixes indicating languages as in `Germany@en` and `Deutschland@de` or data types as in `1^^ http://www.w3.org/2001/XMLSchema#int` and `"1.8542"^^http://www.w3.org/2001/XMLSchema#double`.
As part of the RDF syntax, it is possible to define prefixes such as `dbpedia := http://dbpedia.org/resource/` and then to use them in order to shorten URIs so that `http://dbpedia.org/resource/Darmstadt` becomes `dbpedia:Darmstadt`.

Now we can model information in form of `<subject predicate object>` triples, for example:
`< dbpedia:Merck_KGaA dbpedia-owl:locationCity dbpedia:Darmstadt >`
This triple is supposed to express that the company Merck is located in the city of Darmstadt. Technically, it merely states that a resource with the URI `dbpedia:Merck_KGaA` is in a relationship, identified by the URI `dbpedia-owl:locationCity`, with another resource, represented by the URI `dbpedia:Darmstadt`. Any further meaning is inferred by us humans from the descriptive URIs but (given only this single triple) is not available as such to a computer. This information is codified for the Semantic Web in further triples:
`< dbpedia:Merck_KGaA rdf:type dbpedia-owl:Company>` ("Merck is a company")
`< dbpedia:Merck_KGaA dbpedia-owl:industry dbpedia:Pharmaceutical_industry>` ("Merck is part of the pharmaceutical industry")
`< dbpedia-owl:locationCity rdfs:range dbpedia-owl:City>` ("The range(= possible types of the object) of `locationCity` is city")
`< dbpedia:Darmstadt dbpedia-owl:country dbpedia:Germany>` ("Darmstadt is located in Germany")
This demonstrates the way resources are linked to each other like websites on the WWW.
Only the object of a triple may be a literal, the subject and object must be URIs. This is meant to foster the web structure of Linked Data by allowing users to obtain further information on all subjects and predicates.
An example of a literal in a triple would be:
`< dbpedia:Darmstadt dbpedia-owl:populationTotal 141471(xsd:integer)>`
("Darmstadt has a total population of 141471")

2.1.2 Ontologies

Ontologies describe conceptual knowledge about domains, entities and their relationships in a structured way. The idea of ontologies is borrowed from classical philosophy and can be traced to ancient Greek philosophers. With regards to the Semantic Web, there are various languages for describing ontologies but we will focus on the most prominent ones: RDF Schema (RDFS) and the Web Ontology Language (OWL).

By naming the relationships between resources or between resources and data we have only shifted the problem that computers cannot properly understand strings of text to a higher level. Instead of a single string that it cannot understand, the computer now has to deal with triples of strings that it cannot understand.
However, if we formalize our knowledge of a domain and its classes and their properties, the triples can actually acquire a meaning beyond the three strings they consist of.

For example there is not a lot a machine could do with a triple like `<:Berlin :capitalOf :Germany>` as such because it lacks an understanding of the intrinsic meaning of `capitalOf`. It would not be able to infer seemingly trivial facts such as Berlin being a City and Germany being a country. To do this we would have to explicitly state that the domain, i.e. all subjects of `capitalOf`, are cities and the range, i.e. all objects of `capitalOf`, are countries. Then this deduction would be possible.

RDFS

RDF Schema (RDFS) provides a basic vocabulary for describing ontologies in RDF (Brickley, Guha, & McBride, 2004). It consists of a set of classes and properties.

The classes are:

`Class`: defines an abstract class

`Resource`: the super class of everything (akin to "Object" in Java)

`Property`: the super class of all properties

`Literal`: the super class of all literals

The properties are:

`type`: declares a resource to be an instance of a class

`subClassOf`: a transitive property used to create class hierarchies

`subPropertyOf`: a transitive property used to create property hierarchies

`domain`: declares the type of the subjects of a property

`range`: declares the type of the objects of a property

The prefix for RDFS is `rdfs`. In RDF `a` is used as an alias for `type` as in `<City a rdfs:Class>.`

A sample ontology built with RDFS could look like this:

```
:Place a rdfs:Class
:City rdfs:subClassOf :Place
:Village rdfs:subClassOf :Place
:hasPopulation a rdfs:Property
:hasPopulation rdfs:domain :Place
:hasPopulation rdfs:range xsd:nonNegativeInteger

:Darmstadt a :City
:Geldern-Kapellen a :Village
:Darmstadt :hasPopulation 149052^^xsd:int
:Geldern-Kapellen :hasPopulation 2737^^xsd:int
```

This ontology introduces the classes `Place`, `City`, and `Village`, where `City` and `Village` are sub-classes of `Place`. `Darmstadt` is an instance of a `City`, `Geldern-Kapellen` is an instance of a `Village`. It also introduces the property `hasPopulation` with the domain `Place` and the range `non-negative integer`. Then follow two basic RDF triples stating the populations of the two places.

OWL

The Web Ontology Language (OWL) extends RDFS to allow building complex ontologies (W3C OWL Working Group, 2012).

In OWL there are classes, properties and instances. Every class is a sub class of `owl:Thing` and thus every instance is an instance of type `owl:Thing`.

OWL allows class definitions to use standard set operations such as union, intersection and complement. Restrictions can be placed on properties, for example requiring a certain number of objects or requiring objects to be from a certain class.
In OWL there are two distinct types of properties: `Data typeProptery` and `ObjectProperty`. `Data typeProperties` have literals of a certain *data type* as objects, while `ObjectProperties` have URIs, i.e. references to other resources as objects.

2.1.3 Linked Open Data

Linked Open Data (LOD) is a notion closely related to the Semantic Web. The term was coined by (Berners-Lee, 2006), who proposed four basic rules for publishing data for the Semantic Web:"

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
4. Include links to other URIs. so that they can discover more things.

"

DBPedia

DBpedia is currently a central hub of the Linked Data Cloud as can be seen on Figure 1: LOD Cloud.

Its goal is to make the data contained in Wikipedia available to the Semantic Web.
The current English version of DBpedia, 3.8, contains "3.77 million things, out of which 2.35 million are classified in a consistent Ontology, including 764,000 persons, 573,000 places (including 387,000 populated places), 333,000 creative works (including 112,000 music albums, 72,000 films and 18,000 video games), 192,000 organizations (including 45,000 companies and 42,000 educational institutions), 202,000 species and 5,500 diseases."
Its ontology consists of 359 classes, 800 object properties and 859 data type properties. (Bizer C. , 2012)
DBpedia scrapes data from infoboxes and other structured elements found in many Wikipedia articles to make it available as Linked Open Data. (Bizer, et al., 2009)

Infoboxes

Alongside the main text, Wikipedia offers so-called infoboxes for many articles. These infoboxes contain semi-structured data about the subject of the article. What makes infoboxes so useful for DBpedia is that they contain information in attribute-value pairs and those pairs are somewhat consistent for subjects of a domain. For example countries will typically have attributes such as "Area", "Population", "Capital", movies have attributes such as "Directed By", "Starring" etc.

This makes infoboxes much more suitable for extracting information for Linked Data than the free text that makes up most of the article. However, the data in those infoboxes is still maintained by hand and neither synchronized nor otherwise linked across the different language versions of Wikipedia. This makes it prone to general errors and poses problems during parsing when, for example, thousands separators are not used consistently.

```
{{Infobox German Location
| Art              = City
| image_photo      = Mathildenhoehe-hochzeitsturm-053.jpg|250px
| imagesize        = 250px
| image_caption    = Mathildenhöhe (Wedding Tower)
| Wappen           = Wappen Darmstadt.png
| image_flag       = Darmstadt flag.jpg
| Regierungsbezirk = Darmstadt
| Bundesland       = Hesse
| Landkreis        = urban
| lat_deg          = 49 | lat_min = 52 | lat_sec=0
| lon_deg          = 8 | lon_min = 39 | lon_sec=0
| Lageplan         = Hessen DA.png
| Höhe             = 144
| Fläche           = 122.23
| Einwohner        = 149052
| Stand            = 2007-06-30
| pop_ref          = <ref>{{cite web |title = Area, population and population
change |url=http://www.statistik-
hessen.de/themenauswahl/bevoelkerung-gebiet/regionaldaten/gebiet-
bevoelkerungsstand-und-vorgaenge/index.html |author = Hessian
Statistical Office |language=in German |accessdate=21 May 2007}}</ref>
| PLZ              = 64283-64297
| PLZ-alt          = 6100
| Vorwahl          = 06151, 06150
| Kfz              = DA
| Gemeindeschlüssel = 06 4 11 000
| LOCODE           = DE DAR
| Gliederung       = 9 boroughs
| Adresse          = Luisenplatz 5<br />64283 Darmstadt
| Website          = [http://www.darmstadt.de/ www.darmstadt.de]
| Bürgermeister    = Jochen Partsch
| Bürgermeistertitel = Lord Mayor
| Partei           = [[Bündnis 90/Die Grünen]]
}}
```

Figure 2: Darmstadt Infobox Source

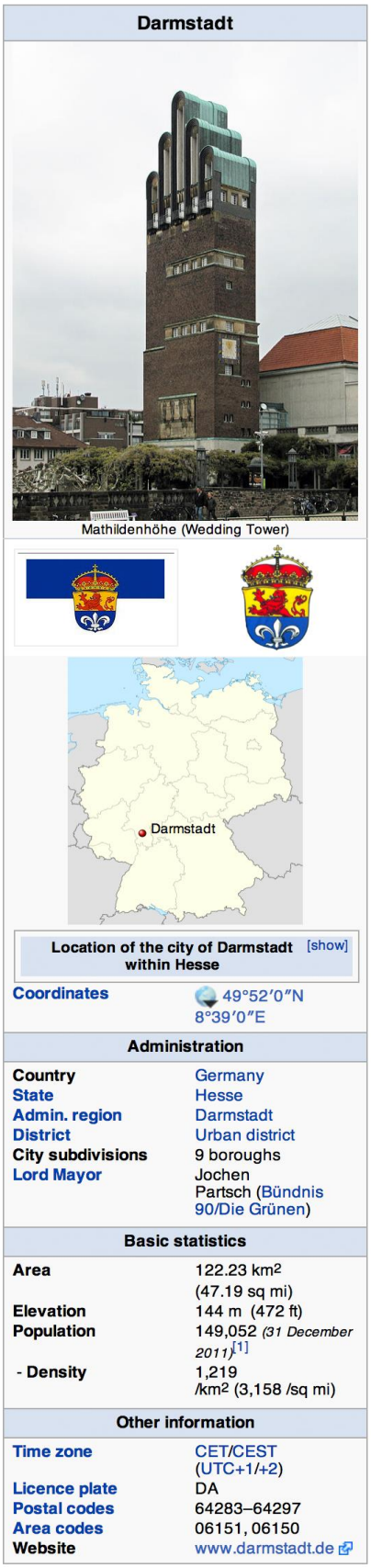

Figure 3: Darmstadt Infobox

SPARQL

SPARQL is a query language for RDF. RDF by itself only allows us to model information and RDF/XML enables us to serialize it but neither provides means to properly query the knowledge graph we have built this way. SPARQL is designed to do exactly that. Its syntax is loosely based on that of SQL. A simple SPARQL query to find the capital of Germany could be "`select ?o where { dbpedia:Germany dbpedia-`

`owl:capital ?o}`. SPARQL queries are run against a SPARQL endpoint; in our case that will be the one of DBpedia, `http://dbpedia.org/sparql`. The `?o` in the query indicates a variable and `dbpedia:Germany dbpedia-owl:capital ?o` corresponds to an RDF triple. The endpoint will then try to find valid bindings for the free variable, `?o`, in the query and return those. It is possible to have multiple variables as in this query to find all countries with their respective capitals: `select ?s ?o where { ?s dbpedia-owl:capital ?o }`. In order to create more complex queries multiple triple patterns can be joined as in `select ?s ?o where { ?s dbpedia-owl:capital ?o . ?s a yago:EuropeanCountries}`. Now the variables must match both triple patterns and thus only countries and capitals in Europe will be returned.

Problems

Since the term "Semantic Web" was coined in 2001, RDF/XML was standardized in 2004 and DBpedia started in 2007, some time has passed and the Semantic Web has still not evolved into an integral part of the Web 3.0, as it was once held out in prospect by Tim Berners-Lee (Shannon, 2006). One problem is certainly the classic catch-22 that many new technologies suffer from: with the existing Linked Data lacking in quantity and quality, there is little incentive to create applications that use it and without applications, that make use of Linked Data, content providers have little incentive to make their data available to the Semantic Web.

DBpedia tries to solve this problem by scraping already existing data from Wikipedia but in doing so deviates from the idea of publishing original data for the Semantic Web. As a consequence of this scraping approach, the data is prone to errors.

2.2 Statistics

2.2.1 Outlier Detection

A commonly found definition of an outlier with regards to statistics is:

"An outlier is an observation which deviates so much from the other observations as to arouse suspicions that it was generated by a different mechanism." (Hawkins, 1980)

The background here is that when we are talking about outliers, we often assume the valid observations to follow some known distribution. Regarding physical experiments and in many other cases, the underlying distribution is frequently assumed to be the normal distribution. E.g., if we are measuring the size of a certain kind of flowers, the underlying mechanism, that influences the distribution of the sizes, is the natural growth of the plants. If, after a significant number of observations has been collected, this distribution is found to be normal but, later on, we discover a specimen that deviates by 10 standard deviations from the mean of the distribution, we could assume that the initial assumption of a normal distribution was wrong and this species in fact varies greatly in size. However, such an extraordinary observation does "arouse suspicions that it was generated by a different mechanism". Namely, that the mechanism that caused this unusual size was in fact not the natural growth of the plant but could possibly be a misidentification of the species on part of the observer.

This fundamental assumption of a specific underlying distribution, however, is problematic with regards to the data we will be dealing with in this thesis. Thus, the following, more general definition could be more useful for us: "An outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs." (Grubbs, Procedures for Detecting Outlying Observations in Samples, 1969)

Outlier detection methods are distinguished by the amount of available training data (Chandola, Banerjee, & Kumar, 2009). Training data are sets of data points that are explicitly labeled as either

normal or outliers. The algorithm can then make use of that data to learn classes of normal and abnormal data points before analyzing the actual data at hand.
There are three main scenarios:
Supervised: training data is available for normal and abnormal cases
Semi-supervised: training data is available only for either normal or abnormal cases
Unsupervised: no training data is available
There are methods for use with one-dimensional or univariate data, where each data point consists only of a single value, and for multi-dimensional data, where each data point is a vector of multiple values.
Data points can either be binary labeled as normal/outlier or given an outlier score that represents the confidence in that point being an outlier, either in a normalized way, i.e. a value from 0.0 to 1.0, or monotonically increasing, i.e. a greater value indicates a greater confidence but does not make direct statements about absolute probabilities.
Given an outlier score, it is generally possible to obtain a binary label by applying a threshold.

Classical Methods
Outlier detection methods were originally developed to define objective criteria for the rejection of abnormal observations in experiments. Prior, this was done at the discretion of the experimenter: "Geometers have, therefore, been in the habit of rejecting those observations which appeared to them liable to unusual defects, although no exact criterion has been proposed to test and authorize such a procedure, and this delicate subject has been left to the arbitrary discretion of individual computers. The object of the present investigation is to produce an exact rule for the rejection of observations, which shall be legitimately derived from the principles of the Calculus of Probabilities." (Peirce, 1852).
Classical outlier detection methods assume an underlying distribution (usually a normal distribution) and then use that assumption to identify outliers: "It should be pointed out that almost all criteria for outliers are based on an assumed underlying normal (Gaussian) population or distribution." (Grubbs, Procedures for Detecting Outlying Observations in Samples, 1969).
For a normal distribution, 99.7% of all values lie within three standard deviations of the mean, so a simple outlier detection approach could be to calculate the distance from each point to the mean and declare each point, that deviates by more than three standard deviations, an outlier.
Various outlier detection schemes, based on similar premises, have been devised over time, e.g. (Peirce, 1852), (Chauvenet, 1863), (Dean & Dixon, 1951) and (Grubbs, Sample Criteria for Testing Outlying Observations, 1950).
However, those methods are unsuitable for our purposes because the data we are dealing with differs greatly from what those tests were designed for. Most importantly, the assumption of a normal distribution is not compatible with the vast range of different datasets found in DBpedia and unsupervised approaches. Further problems arise with regards to methods being designed for small sample sizes, e.g. (Dean & Dixon, 1951), or only being able to detect one or a few outliers at a time, e.g. (Grubbs, Sample Criteria for Testing Outlying Observations, 1950).

General Problems
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 manner. For example, if all regular ZIP codes have 7 digits it should be possible to detect invalid ZIP codes with 3 or 13 digits. However, if a ZIP code is simply wrong as in "543892" instead of "592637", it will be practically impossible to detect as an outlier without flagging large portions of correct ZIP codes as outliers.

On the other hand, some data points are certainly outlying but nonetheless correct: Pauline Musters (`http://dbpedia.org/resource/Pauline_Musters`) was at 58 centimeters the shortest adult woman ever recorded, which certainly makes here an outlier regarding her size but does not mean the value of 0.5842m, given at DBpedia for her height, is incorrect. Other cases are less obvious, for example, Muggsy Bogues (`http://dbpedia.org/resource/Muggsy_Bogues`) is, at 1.6 meters, shorter than the average man but would probably not be identified as an outlier among the general population. However, among basketball players, he certainly represents an outlier.
Similarly, states such as China and India with populations of over one billion are outliers between the millions of towns and villages with only a few hundred or thousands of people.
Therefore, we do not understand outlier detection as an objective measure of the correctness of data but rather as a means of identifying suspicious data points that should be inspected by hand.

Figure 4: Pauline Musters at around age 19, next to a man of average height

2.2.2 Robust Statistics

Robust statistics are an important concept when dealing with outliers. Traditional statistical measures can be extremely sensitive to outliers whereas robust statistical methods are supposed to perform reasonably well, even when assumptions are not exactly met. We will now look at estimator for two important statistical properties.

Central Tendency

The central tendency of a distribution is a central value of the population that gives an idea of where it is located. Typical measures of the central tendency are the mean ($\frac{1}{n}\sum_{i=1}^{n} a_i$), the median (the point in the middle of the sorted list of the data points) and the mode (the single most common value).
The mean of [1, 1, 3, 3, 4, 5, 5, 5, 6, 7] is 4 and, as would be expected, marks roughly the middle of the population. However, if a single observation is erroneous, the mean can assume virtually arbitrary values. If the 4 were somehow misinterpreted as 4,000, the mean would then become 403.6, which is nowhere near the middle of the population. If we were to use the median, we would get a value of 4.5 for the original population and of 5.0 for the corrupted population of [1, 1, 3, 3, 5, 5, 5, 6, 7, 4000], which is still a good estimate of the central tendency of the population. In fact, we could replace almost up to half of the population with arbitrary values and still get a reasonable estimate: The median of [1, 3, 5, 5, 5, 6, 1000, 3000, 4000, 7000] is 6, as opposed to the mean of 1502.
If we add a fifth erroneous value, the median would finally assume arbitrary values too. We call this behavior the breakdown point of the estimator: the proportion of incorrect values that are needed to make the estimator yield arbitrary results. As we have seen, this value is 0 for the mean, because it cannot handle a single incorrect value without "breaking down" and 0.5 for the median, because we can replace half the population before it breaks down. This also happens to be the highest possible breakdown point as there would be no way to determine which observations are supposed to be "regular", when half or more of the population is erroneous.

Dispersion
Another important property of a statistical population is its dispersion. The dispersion is a measure of how close together or stretched over a wide range observations are.
The traditional measure of dispersion is the variance or the directly related standard deviation. The standard deviation of a sample population is frequently estimated using the sample standard deviation $\sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$. This estimator is not robust however, as it uses the mean, which we have already seen to be heavily affected by outliers, and squares of the distances of each point to that mean, thus giving more weight to outliers than to regular observations.

The range, the difference between the largest and the smallest observation, is another measure of dispersion, which too is obviously extremely affected by outliers. To mitigate this, we can use the range of a cropped sample, which leads us to the interquartile range.

Interquartile Range
The quartiles of a population are the 1-quartile (=Q_1) below which we find 25% of the population, the 2-quartile (=Q_2=median) below which we find 50% of the population, the 3-quartile (=Q_3) below which we find 75% of the population and finally the 4-quartile (=Q_4), which includes the whole population. The interquartile range (IQR) is then defined as $Q_3 - Q_1$ and gives a rough measure of the spread of the population, while being robust to outliers because outliers, traditionally being the largest or smallest observations, are not taken into account. The breakdown point of the IQR is 0.25 because if we replace the largest 25% of the population with large values, the next value we add will be taken into account for the IQR.

MAD
The Median absolute deviation (MAD) is a classic (the concept is already mentioned in (Gauss, 1816)) example of a robust measure of dispersion. It is defined as MAD = $median_i\left(\left|X_i - median_j(X_j)\right|\right)$.
This means, 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.
For example, given a list of observations such as (3, 7, 15, 3, 4, 2, 1000, 5, 9), we first sort it (2, 3, 3, 4, 5, 7, 9, 15, 1000) to obtain the median of 5. We then create the list of all absolute deviations from that median (3, 2, 2, 1, 0, 2, 4, 10, 995) and sort again (0, 1, 2, 2, 2, 3, 4, 10, 995) to get the end result of 2.
One can intuitively see the robustness of this measure in the double use of the median, which makes it robust to outliers when estimating the center of the population and when estimating the center of the deviations.

S_n/Q_n/scaleTau2
S_n and Q_n are dispersion estimators proposed by (Rousseuw & Croux, 1993) to improve upon the MAD. While the MAD is extremely robust, it is aimed at symmetric distributions and has subpar efficiency for actual Gaussian distributions. Both are based on pairwise differences of all points as opposed to differences to the median or between two single points.

S_n is defined as $c \cdot median_i\{median_j|x_i - x_j|\}$. This means, we calculate for each point the distance to each other point and take the median of those values. This yields n results, the median of which will be used as the end result of the estimator. The factor c is meant to create consistency so

that the dispersion estimate for a standard normal distribution will converge against 1 and will be chosen as 1.1926.
The motivation behind this algorithm is to make it independent of the estimation of a center, so it will work with asymmetric distributions as well as with symmetric ones.

Q_n is defined as $d\left\{|x_i - x_j|; i < j\right\}_{(k)}$ where d is again a constant factor used for consistency and $k = \binom{n/2 + 1}{2} = \frac{n}{4}\left(\frac{n}{2} + 1\right)$. That means we calculate the pairwise distances for all points but take into account symmetry, so the distance between a and b will only be recorded once. We then order those distances and take the kth order statistic of those values as the end result. Again, the pairwise distances provide independence from location estimates but Q_n features a breakdown point of 0.5 as opposed a breakdown point of (n/2)/n for S_n.

ScaleTau2 is an estimator proposed by (Maronna & Zamar, 2002). In order to define scaleTau2, we need to define some auxiliary functions first. Let X be our random sample of $x_1, \dots, x_n$.

$$W_c(x) = \left(1 - \left(\frac{x}{c}\right)^2\right)^2 I(|x| \leq c) \; and \; \rho_c(x) = \min(x^2, c^2).$$

ScaleTau2 works in two steps. First we calculate $\sigma_0 = MAD(X)$ and a set of weighting functions

$$w_i = W_{c_1}\left(\frac{x_i - median(X)}{\sigma_0}\right)$$

Next, we estimate the location

$$\mu(x) = \frac{\sum_i x_i w_i}{\sum_i w_i}$$

And finally the scale

$$scaleTau2 = \sigma(x)^2 = \frac{\sigma_0^2}{n} \sum_i \rho_{c_c}\left(\frac{x_i - \mu(X)}{\sigma_0}\right)$$

2.2.3 Kernel Density Estimation

Histograms provide for a simple and intuitive way to approximate an underlying density function based on a number of observations. However, since the resulting function is essentially a sum of rectangle functions, it is decidedly discontinuous and if we assume the data to be generated by a natural process, a continuous function could be more desirable. Kernel Density Estimation addresses this shortcoming by replacing the rectangle functions with a so-called kernel K, a symmetric, non-negative function that integrates to 1. If $x_1, x_2, \dots x_n$ are independent and identically distributed (iid) random variables, drawn from a distribution with an unknown density function f, then

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

is its kernel density estimator.
(Rosenblatt, 1956) and (Parzen, 1962) are credited with independently developing this method.
For our purposes, we will use the Gaussian normal distribution, $\frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$, which satisfies all requirements of a kernel. The bandwidth h can be chosen according to "Silverman's rule of thumb" as $\left(\frac{4\hat{\sigma}^5}{3n}\right)^{\frac{1}{5}}$, where $\hat{\sigma}$ is the sample standard deviation. This bandwidth yields optimal results for cases

where the underlying distribution is actually normal and reasonable results for unimodal, symmetric distributions (Härdle, Müller, Sperlich, & Werwatz, 2004).
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, $mp = \frac{1}{n}\sum_{i=1}^{n} \widehat{f_h}(x_i)$. The relative probability of one data point being normal is then $\mathrm{rp(x)} = \frac{\widehat{f_h}(x)}{mp}$.
rp(x) > 1 indicates an above average probability, rp(x) < 1 indicates a below average probability. To obtain a binary label, we can apply a threshold, e.g. all x with rp(x) < 0.1 could be considered outliers.

2.2.4 Clustering

Clustering is the task of grouping data points in such a way that the data points in each cluster are in some manner similar to each other. There is no definite measure of this similarity and what is considered similar may change from application to application (Estivill-Castro, 2002).

Expectation–maximization Clustering

Expectation-maximization (Dempster, Laird, & Rubin, 1977) is an iterative algorithm that can be used for clustering.
Given a set of data points, the goal is to find parameters for clusters and determine membership of each data point for one of those clusters, so that the overall likelihood of the model is optimized.
The algorithm is initialized by guessing parameters for the model. It then iterates in two steps:
Step 1, Expectation: Compute for each data point the probability of membership in each cluster according to the current set of parameters
Step 2, Maximization: Vary the parameters so that the probabilities of step 1 are maximized
Those two steps are repeated until the algorithm converges, i.e. no more change is observed, or a certain limit of iterations is reached.

2.3 Related Work

(Zaveri, et al., 2012) created a taxonomy of errors in LOD and proposed a crowd-sourcing approach for assessing the quality of DBpedia.
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.
For the crowd-sourcing assessment, a toolkit, *TripleCheckMate*, was developed that allows users to sign up and inspect resources one by one. A competition was started, where the most diligent and successful participants could win a prize. 58 users evaluated a total of 521 resources and identified 2,928 erroneous triples. Based on the data collected this way, 11.93% of all triples were found to be affected by some kind of error, though most were deemed fixable through changes to the extraction process.

(Fleischhacker, 2013) studied numerical error detection in Linked Data using KDE on the properties `birthDate`, `birthYear`, `deathDate` and `deathYear` datasets. Fleischhacker successfully used the correlations between `birthDate` and `deathDate` and between `birthYear` and `deathYear` to improve results. Using this method, the top-50 outliers were all found to be in fact incorrect.

3. Approaches

Our approaches are based on analyzing the subjects and literal values for a given predicate, that is the results of the SPARQL query `select ?s ?o where {?s predicate ?o}`. This means we are dealing with tables of triples such as `<http://dbpedia.org/resource/Darmstadt dbpedia-owl:populationTotal 141471(xsd:integer)>`. The data, that we are analyzing, will be the literals in the object column, `141471(xsd:integer)` in this case. However, this data is not necessarily available in a consistent, numeric form, which leads us to our first approach.

3.1 Data Types

A simple yet effective approach to detect faulty data is to analyze the data types of the literals. Many properties do not have a declared range and their objects will then often be of various data types. The idea is that most numerical properties will have a certain intended range and literals of that range will make up the majority of its objects. For example, a population count should be represented by an integer and the height of a person by a double value. If 99% of all values for a certain predicate share a single type then it seems fair to assume that the remaining 1% is erroneous in some way.

In many cases numerical properties have literals that hold the correct value but in string form, for example `"-8"@en` instead of `"-8"^^xsd:int`. This would prevent a regular user from accessing the value of the literal because an integer would be expected. Once those triples have been identified, it should be easily possible to correct the errors by simply parsing the existing strings.

3.2 Basic Outlier Detection

We will now look into basic methods to identify outliers in numerical data. What distinguishes these simple approaches from the following ones is that they do not take into account the semantics that are available on the Semantic Web. We merely analyze the numerical values of all objects in a dataset to identify outliers. Thus, the data is one-dimensional and this allows us to sort it, use the median and yields a simple distance measure.
As mentioned earlier, we are looking at an extremely diverse set of data and this means we cannot safely make a lot of assumptions. The main assumption we are going to make here is that the data is unimodal to some extent. A distribution is unimodal if it consists of a single peak, i.e. the density function f(x) has one maximum, f(m), and is monotonically increasing for all x < m and monotonically decreasing for all x > m. This is a minimum requirement for most of the following methods and generally seems reasonable. To verify this assumption, we created histograms (see appendix, "Res50 histograms") for the datasets created from the 50 random resources discussed later on. Visual inspection showed that indeed of the 170 datasets, 154 (90.6%) were clearly unimodal.

3.2.1 Interquantile Range

Given the interquartile range (IQR) and some factor k, we can then label outliers as follows: Every point $p \in [-\infty, Q_1]$ with $Q_1 - p > k \cdot (Q_3 - Q_1)$ is an outlier, every point $p \in [Q_3, \infty]$ with $p - Q_3 > k \cdot (Q_3 - Q_1)$ is an outlier and every other point is considered normal.

This approach can then be generalized to quantiles where the population is split into an arbitrary number of equal parts. For example, if we split the population into 100 parts, then called "percentiles", we can use the range between the 5-percentile (Q_5), below which 5% of the population is found, and the 95-percentile (Q_{95}), below which 95% of the population is found, as a measure of dispersion.

We will call this method “IQR”.

3.2.2 Dispersion and Center

A similar approach is to use an estimate for the central tendency and combine this with a general measure of dispersion to define the range of normal values.
We used the median to estimate the center of the population. As dispersion estimators, we used MAD, Sn, Qn and ScaleTau2. The range of normal values is then defined as $median \pm k \cdot dispersion\ estimate$.
We will call this family of methods “dispersion”.

3.2.3 Kernel Density Estimation

These two approaches have obvious limitations in that they can only detect outliers among the largest and smallest data points. KDE is more flexible because it does not deem one interval normal and everything else an outlier.

We tested three implementations for generating KDE’s. The ones found in Weka (The University of Waikato, 2013) and SSJ (L'Ecuyer, 2012) allow creating a KDE and then querying the probabilities of arbitrary points. The drawback is that this method is rather slow for large datasets. The implementation provided by R uses fast Fourier Transform (FFT) to achieve extreme performance gains but this requires the samples to be equidistant and their number to be a power of two. Up to about one million samples is feasible so we either have to restrict ourselves to smaller ranges or suffer accuracy losses. The obvious drawback of this approach is that it is extremely affected itself by outliers, so a sensible implementation could be to first use a more robust method to remove extreme outliers and then use KDE(FFT) to screen for more subtle ones.
We will use “KDE” and “KDE-FFT” to refer to the respective methods.

3.3 Iteration

All the methods above can be used iteratively. That is, we apply the method to a dataset, remove all data points that were labeled outliers and then repeat the process until no more outliers are found. Alternatively an upper limit for the number of iterations can be set. This works because outliers gravely affect some statistical metrics, thus removing them allows us to more closely analyze the remaining data points.

3.4 Semantic Outlier Detection

The simple outlier detection approach is limited by the existence of natural outliers. Consider a property such as *populationTotal* (`http://dbpedia.org/ontology/populationTotal`), which represents the total population of a `populatedPlace` (`http:// dbpedia.org/ ontology/PopulatedPlace`). This includes villages, towns, cities, states, countries, continents and – contrary to the label – some unpopulated places such as ghost towns and uninhabited islands. That means China, India 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 most of the entries, by far.
However, if we analyze the subjects solely with regards to other subjects of their kind, e.g. examining the population of towns alone, this problem could be overcome. The semantics of the Linked Data allow us to do exactly that.

3.4.1 Splitting by Single Type

In RDF(S), resources can be instances of a class, i.e. have a certain type and , as can be seen on Figure 5: Type Count Frequencies, many resources on DBpedia do have one or multiple types. We can utilize those types to separate the subjects of a predicate into groups and then further analyze them.

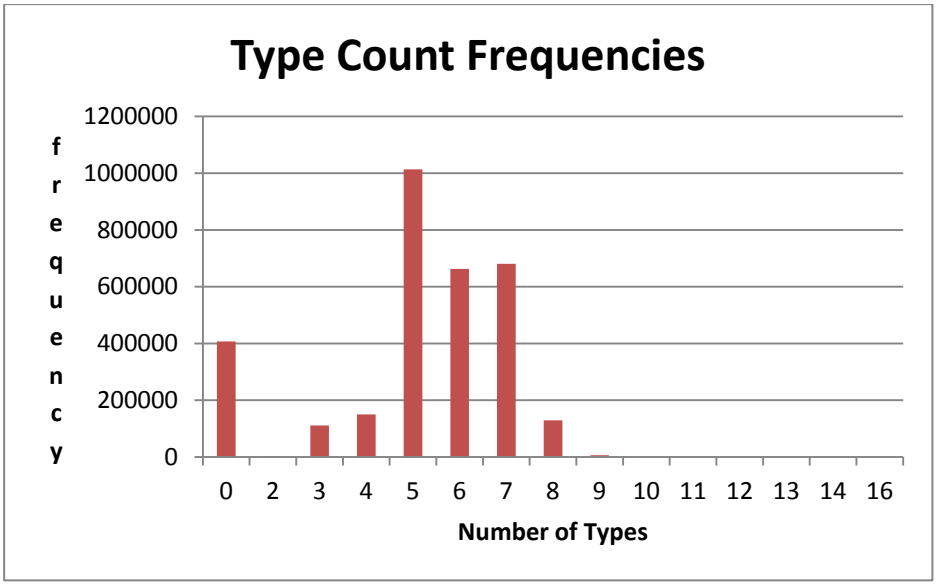

Figure 5: Type Count Frequencies

The obvious approach is to split the dataset into one subset for each type. To do this we have to select and filter certain types.

This is the set of `rdf:types` for Darmstadt. We will use those to exemplarily identify strategies for selecting useful types:

```
owl:Thing
dbpedia-owl:Place
gml:_Feature
dbpedia-owl:City
http://schema.org/Place
dbpedia-owl:PopulatedPlace
dbpedia-owl:Settlement
http://schema.org/City
http://umbel.org/umbel/rc/City
yago:YagoGeoEntity
http://umbel.org/umbel/rc/Village
http://umbel.org/umbel/rc/Location_Underspecified
http://umbel.org/umbel/rc/PopulatedPlace
```

Generic Types

Because 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, so when clustering by simple types it is advisable to check first if the cluster represents a proper subset.

Type Hierarchies

When we look at the types `dbpedia-owl: Place`, `dbpedia-owl: City`, `dbpedia-owl: PopulatedPlace`, `dbpedia-owl: Settlement` and the DBpedia ontology, we can see that they all form part of one hierarchy, where `dbpedia-owl: City` < `dbpedia-owl: Settlement` < `dbpedia-owl: PopulatedPlace` < `dbpedia-owl: Place` < `owl: Thing`. In order to only make use of the most specific types, we can filter out all those types that are a super class of another type in the set.

Faulty Types

With `http://umbel.org/umbel/rc/Village`, we can see one fundamental problem of this approach. With a population of almost 150,000 Darmstadt is by every common definition and the definition given at umbel.org, "A subcollection of City. Each instance of Village consists of a small cluster of buildings located in a rural area. Most of the buildings are typically residents' houses; others might include shops, schools, churches, and the like. Many residents of a village typically work in the surrounding coutryside (say as farmers). Since Village is a subcollection of City (q.v.), an instance must in some way be self-governed." (Village.rdf), not a village. However since "Anyone Can Say Anything About Anything " (Klyne, Carroll, & McBride, 2004) regarding Linked Data, it still appears as such in DBpedia. This is actually not an outlier in itself either, as can be seen with the SPARQL query:

```
select ?s ?o where {?s a <http://umbel.org/umbel/rc/Village> . ?s dbpedia
-owl:populationTotal ?o filter(?o > 1000000)} order by desc(?o)
```

There are 290 "villages" with a population of over 1 million, including Shanghai, Tokyo, Mexico City and many other mega cities.

That means we have to rely on possibly faulty data in order to fix other possibly faulty data.

To mitigate this problem, we can filter the types by prefixes to only allow types from certain sources. For example we could limit the types to those from DBpedia only.

But even in the DBpedia ontology, there are villages with populations of over 100,000 (`http://dbpedia.org/resource/Lassan_Thakral`, `http://dbpedia.org/resource/Chandanaish_Upazila`, `http://dbpedia.org/resource/Brgat`).

On the other hand there are "cities" with less than ten inhabitants such as `http://dbpedia.org/resource/Warm_River,_Idaho`, `http://en.wikipedia.org/wiki/Lakeside,_Missouri` and `http://en.wikipedia.org/wiki/Ruso%2C_North_Dakota`.

Similarily, the classification of the 7.4 meter high ultra class vehicle, "Liebherr T 282B" (`http://dbpedia.org/resource/Liebherr_T_282B`) as `dbpedia-owl:Automobile` appears debatable and differs at the very least greatly from what one would usually imagine for an automobile.

Figure 6: Liebherr T 282B - 7.4 meter high "automobile", photo by René Engel

Ultimately, we do not have much choice because if we limit the range of possible types too much, there will not be much left to correct.

3.4.2 Clustering by Type Vectors

Since we have seen isolated types to be misleading and as can be seen on Figure 5: Type Count Frequencies, most subjects feature a multitude of types, the natural next step is to utilize the combined information of all types.
For this approach we consider the types of a subject as a vector of Booleans, representing whether or not the subject is of a certain type, and then apply traditional clustering techniques to this vector. To create these vectors, we used FeGeLOD (Paulheim & Fürnkranz, 2012).

FeGeLOD (Feature Generation from Linked Open Data) is a toolkit that is designed to automatically enrich resources with information gathered from Linked Open Data. There are various feature generators available but we used the type generator that takes a set of input subjects and generates a type vector for each subject. FeGeLOD first collects the information for all subjects and then applies a threshold p to remove 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 by WEKA (The University of Waikato, 2013), using the Estimation-Maximization (EM) method.

3.5 Implementation Detail

The implementation was done in Java with major use of the Jena libraries for communication with DBpedia and handling of the Linked Data. Other libraries used include WEKA (The University of

Waikato, 2013) and FegeLOD (Paulheim & Fürnkranz, 2012) for the clustering algorithm and SSJ (L'Ecuyer, 2012) for the exact KDE implementation.
The R programming language (The R Project for Statistical Computing, 2013) and its libraries (especially "robustbase") were used for general analysis and integrated into the Java application to utilize the dispersion estimators as well as the FFT implementation of KDE.
A MySQL database was used to store type information and original as well as parsed versions of triples.

3.5.1 Data Representation and Parsing

We used two representations for triples. The first, *RawData*, consists of a predicate and a list of *Rows* that each hold the information about a corresponding subject and object as a Jena node. The objects, such as `"1831.5"^^http://www.w3.org/2001/XMLSchema#`, are not yet available for processing as doubles so we have to parse them first. Since this format is standardized, this can easily and accurately be done.
We also try to parse objects that do not exactly meet the specifications, such as `"1976 m"@en`, to be able to deal with as many triples as possible. To do this, we use the basic Java `numberFormat.parse` method to convert strings to double values. This `numberFormat.parse` method tries to parse a number from the beginning of a string and does not perform much error correction. We experimented with more sophisticated parsing methods but those were prone to false positives, i.e. even parsing strings that were not meant to be interpreted as numerical values such as a text containing some numbers.
We stored the results in *DataSets* consisting of a predicate and a list of *DataPoints* that each hold the subject as a string and the object as a double.

3.5.2 AnalysisMethod and Analyzor

We distinguish between *analysis methods* and *analyzors* or *modes*.

3.5.3 Analysis Method

AnalysisMethod is an interface that represents a basic outlier detection method and consists of a single method, `void setOutlierScores(DataSet dataset)`, that calculates the outlier scores for each data point in a dataset. We chose not to go with a normalized outlier score from 0.0 to 1.0, as it would not have been possible to make meaningful claims about the probability of a data point actually being incorrect without making too many assumptions about the underlying distribution.

3.5.4 Outlier Score

Instead we used a monotonically increasing outlier score where possible. For example, with regards to the *dispersion* methods this would mean that any data point that is farther than $k \cdot estimated\ dispersion$ away from the median would get an outlier score of 1.0 + calculated distance. This outlier score does not make any direct implications about the likelihood of the corresponding data point actually being an outlier but it creates an order, where for any two data points a and b, outlier_score(a) > outlier_score(b) indicates that a is more likely to be an outlier than b, judging by that method. Thus, if we find a large number of possible outliers, we can sort them by the outlier score and then only choose to inspect those with the highest outlier score.

3.5.5 Analyzor

Analyzors represent a mode of operation for detecting outliers. The Analyzor interface consists of a single method `void run(DataSet dataSet, List<DataPoint> resultsIn` where `dataSet` is the dataset to be analyzed and `resultsIn` is a list for the detected outliers to be placed in. Possible modes of operation are:

The *default* mode that runs a given analysis method on the dataset and returns all data points with an outlier score above a certain threshold. This is the only analyzor that directly wraps an analysis method, all other analyzors work only on analyzors themselves.
The *iterative* mode that runs an analyzor, removes the detected outliers from the dataset and then reruns it until no more outliers can be found or a certain limit of iterations is reached.
The *byType* mode that splits the dataset up by RDF types and then analyzes each resulting subset individually.
The *cluster* mode uses the type feature generator of FeGeLOD with a threshold of 0.95 to create instances for WEKA, which are then clustered using the EM algorithm with a maximum of 100 iterations, no set number of clusters to create and a minimum allowable standard deviation of 10^{-6} for normal density calculation. Then each cluster is analyzed individually.

4. Evaluation

We went through three stages to evaluate our approaches. Firstly, we used simulated data to test the implementation and to get a general idea of the performance of each algorithm.
We then chose three properties to find out how well the algorithms perform on real data.
In order to obtain representative results, we finally selected 50 resources at random and used the most successful methods from the last stage on the predicates of those resources.

4.1 Methods

We mainly evaluated the algorithms IQR, Dispersion (MAD, Sn, Qn, ScaleTau2) and KDE-FFT. In order not to introduce more parameters, we regulated the number of outliers through the intrinsic parameters of each method instead of only examining the top n results by outlier score.

4.1.1 IQR

We went with the percentile implementation of the IQR, so the two parameters are the percentile used and the factor k that determines how many times larger than the interquantile range the distance of a data point to the lower or upper percentile has to be in order for that point to be considered an outlier.

4.1.2 Dispersion

We evaluated all four previously introduced dispersion estimators with the sole parameter being the factor k that determines the range of normal values.

4.1.3 KDE

We evaluated both KDE and KDE-FFT with the single parameter being the relative probability threshold. However, since the runtime of KDE proved to be prohibitively high for the main test track, it will be evaluated on its own.

4.2 Modes

We used the modes *default*, *byType*, *cluster* and *iterative* as described in 3.5.5.
For the *byType* mode we limited the total number of subsets to the 50 largest with at least 100 data points. We also only used types from the DBpedia ontology and only those that represent a leaf.

4.3 Test Tracks

We built test tracks that automatically run analysis methods with varying parameters on datasets. To enable automatic evaluation, we built lists of correct and incorrect values by manually examining the results of each run. After each run, we compared the newly detected outliers to the corresponding lists and classified them as either true positive, false positive, false negative or unknown. The list of incorrect (=true positives) values not only includes clear outliers but everything that deviates by more than a rounding error from the correct value. That means a high number of false negatives is generally to be expected.

4.4 Simulated Data

Since we do not have any training data available and accurately labeling any reasonably large data set by hand would be infeasible, we chose to first run tests on simulated data.
We created sample data following a normal distribution. We used the Marsaglia polar method (Marsaglia & Bray, 1964) to generate the sample data.

We used a single normal distribution with a mean of 1,000 to create 5,000 data points and added 50 outliers uniformly distributed in the range of [-1000, 3000]. We created 49 such datasets and increased the standard deviation of the normal distribution from 1 for the first set to 50 for the last set. However, as the "outliers" are placed completely at random, some will fall into the range of valid data points and will thus be practically impossible to detect without causing false positives.

We will now take a look at the Receiver Operating Characteristics (ROC) curves for the simulated data. These curves plot the number of true positives against the number of false positives and give a general idea about how well the methods perform. Ideally we would constantly have zero false positives so we would see an infinite slope. However, for any meaningful analysis, false positives have to occur at some point so the curve will begin to move to the right, but the later that happens the better.

Results are generally as excellent as we would expect them to be for normally distributed data. The ROC curves of all methods are almost rectangular. All except the KDE-FFT method are able to identify over 2,000 outliers without incorrectly flagging a single normal value. At around 2,350 out of 2,450 possible outliers all methods begin to produce large numbers of false positives.

IQR and *dispersion* perform almost identically here, being able to correctly detect 2,301 and 2,300 outliers without a single false positive. Beyond that point, the number of false positives escalates quickly, with the last useful results at 2,357:288 and 2367:530 respectively.

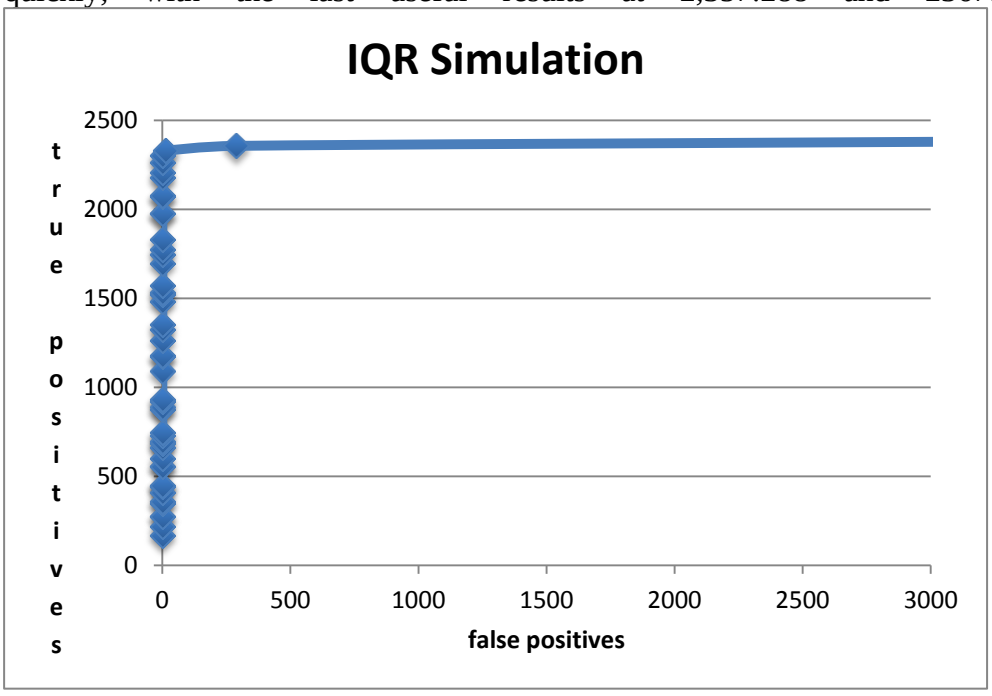

Figure 7: IQR simulation ROC

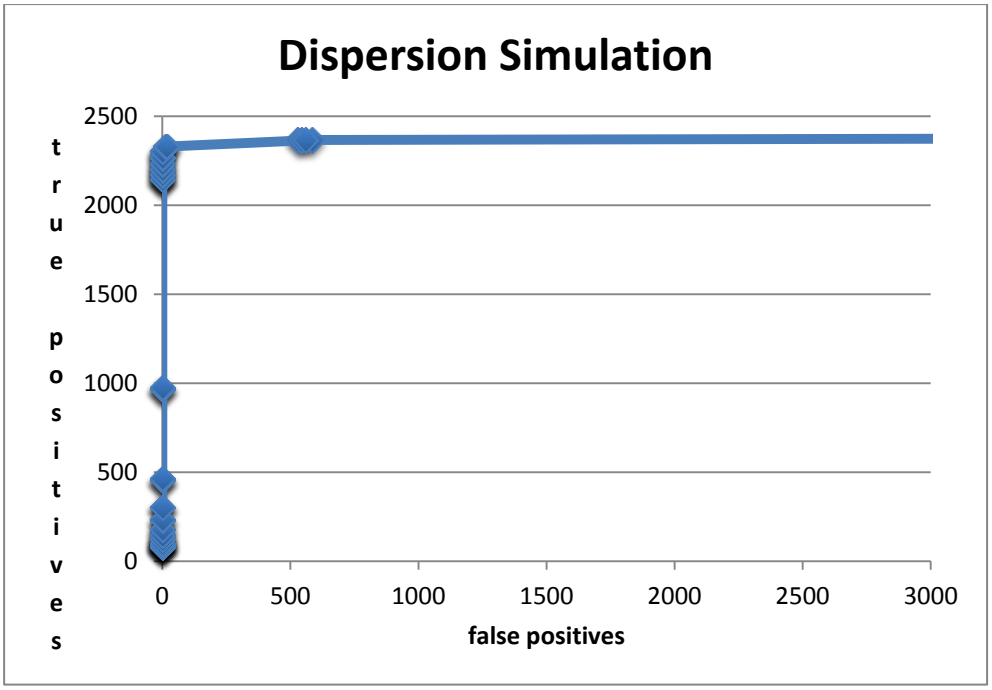

Figure 8: Dispersion simulation ROC

The two KDE version perform slightly worse but overall still excellently. The last error free point is at 2,199 correctly identified outliers for basic KDE. The FFT version accumulates false positives right from the start but only 26 in total at the 1,895 point. Beyond that point, it even slightly outperforms the original version: The FFT version picks up only 159 false positives for 2,347 true positives while the original version finds 2,740 false positives at this point.

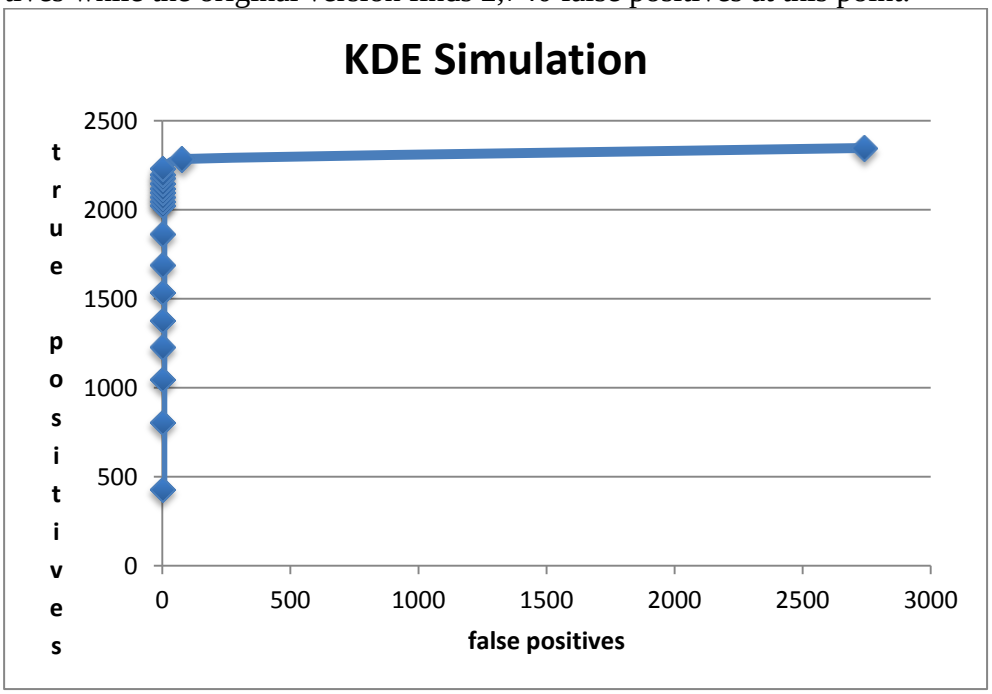

Figure 9: KDE simulation ROC

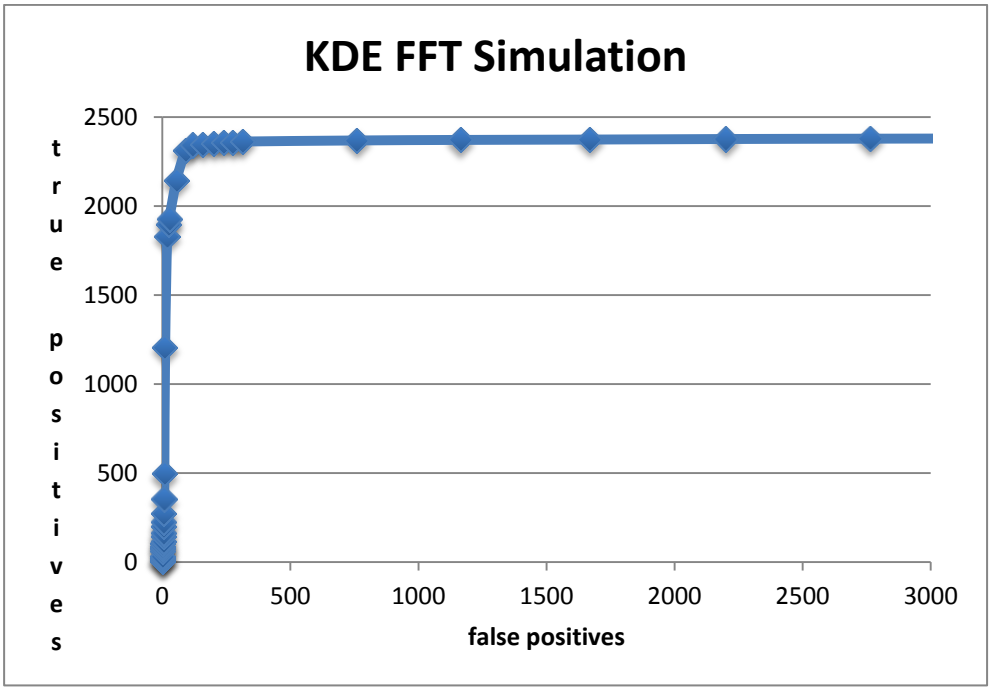

Figure 10: KDE-FFT simulation ROC

Overall, the results are not very surprising or meaningful, merely proving that each of the methods is capable of finding outliers among normally distributed data points. Once we turn to real data, results will be much different, so there is not much point in trying to optimize parameters for the simulated data.

4.5 Specific Predicates

We chose certain predicates for more intensive manual analysis. Those predicates include `dbpedia-owl:populationTotal`, `dbpedia-owl:height` and `dbpedia-owl:elevation`. This choice was based on some interesting properties of those predicates.
We will have a look at the predicates first to better understand the results of our tests.

4.5.1 PopulationTotal

populationTotal has a broad data base of 237,700 triples with entities from various domain such as villages, cities, states, countries, continents and international organizations. This makes it interesting for semantic analysis as on first glance there is only one completely obvious outlier, `http://dbpedia.org/resource/Sisak`, with a population of 476,992,030,567,891 on the other hand, there are various errors such as `http://dbpedia.org/resource/Machchhegaun`, a town in Nepal with an alleged population of 59,510,000 (actually 5,951), which are masked by the existence of real entities with such population numbers. Valid values range from zero for ghost towns to billions for entire continents.
Furthermore, with a quarter million triples *populationTotal* is one of the larger predicates on DBpedia, which makes it suitable as a benchmark for the feasibility of an algorithm.
Our basic parsing methods successfully parsed 237,700 (100.0%) of the triples.

The 0-10,000 inhabitants category comprises the vast majority of the records, however on the other hand of the spectrum there are a few places with billions of people.

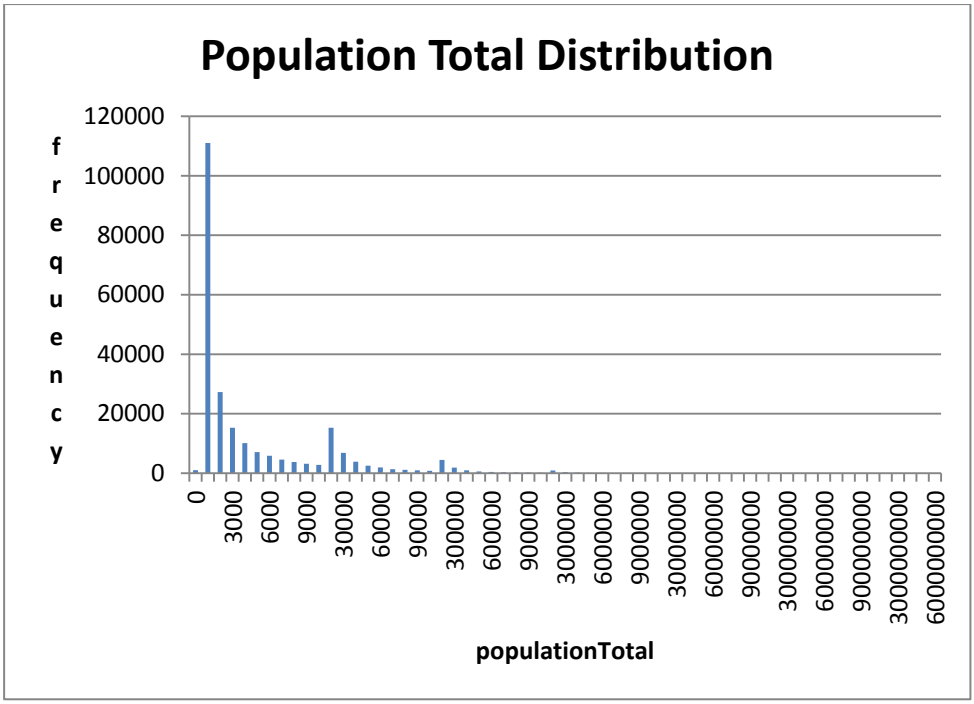

Figure 11: Population Total Distribution

Next, we have the population split up by RDF types as done by our *byType* mode. The distribution is different because not all objects feature the specific types that we are using to split the dataset. Villages make up almost half of the dataset and subjects with population numbers of over 200,000 are sparse.

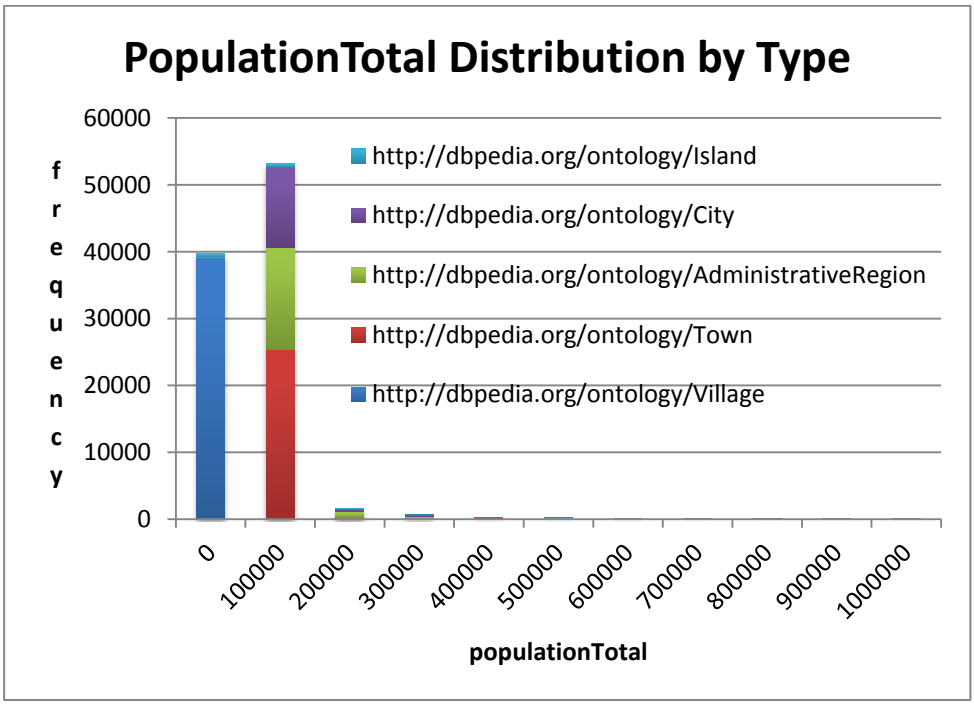

Figure 12: Population Total Distribution by Type

4.5.2 Height

height contains the heights of people (mostly athletes), vehicles, buildings and other objects in meters. It is interesting mainly because of the high number of errors due to unit conversions.
In total there are 52,252 triples, out of which 52,211 (99.9%) were successfully parsed.
Legitimate values range from 0.3 meters for a land mine, `http://dbpedia.org/resource/L10_Ranger_Anti-Personnel_Mine`, to 671 meters for a planned skyscraper in Las Vegas, `http://dbpedia.org/resource/Millennium_Tower_(Las_Vegas)`.

This is the distribution of all objects:

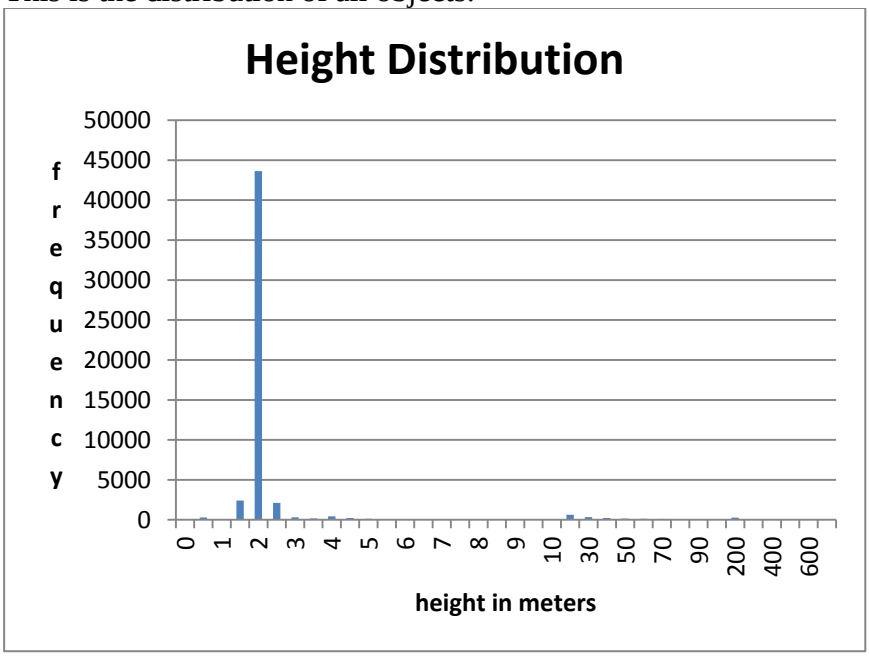

Figure 13: *Height* Distribution

This chart shows the distribution for the eight most common RDF types. We can see that persons, or more specifically athletes, make up the vast majority of the subjects and only lighthouses contribute to the range beyond the five meters mark.

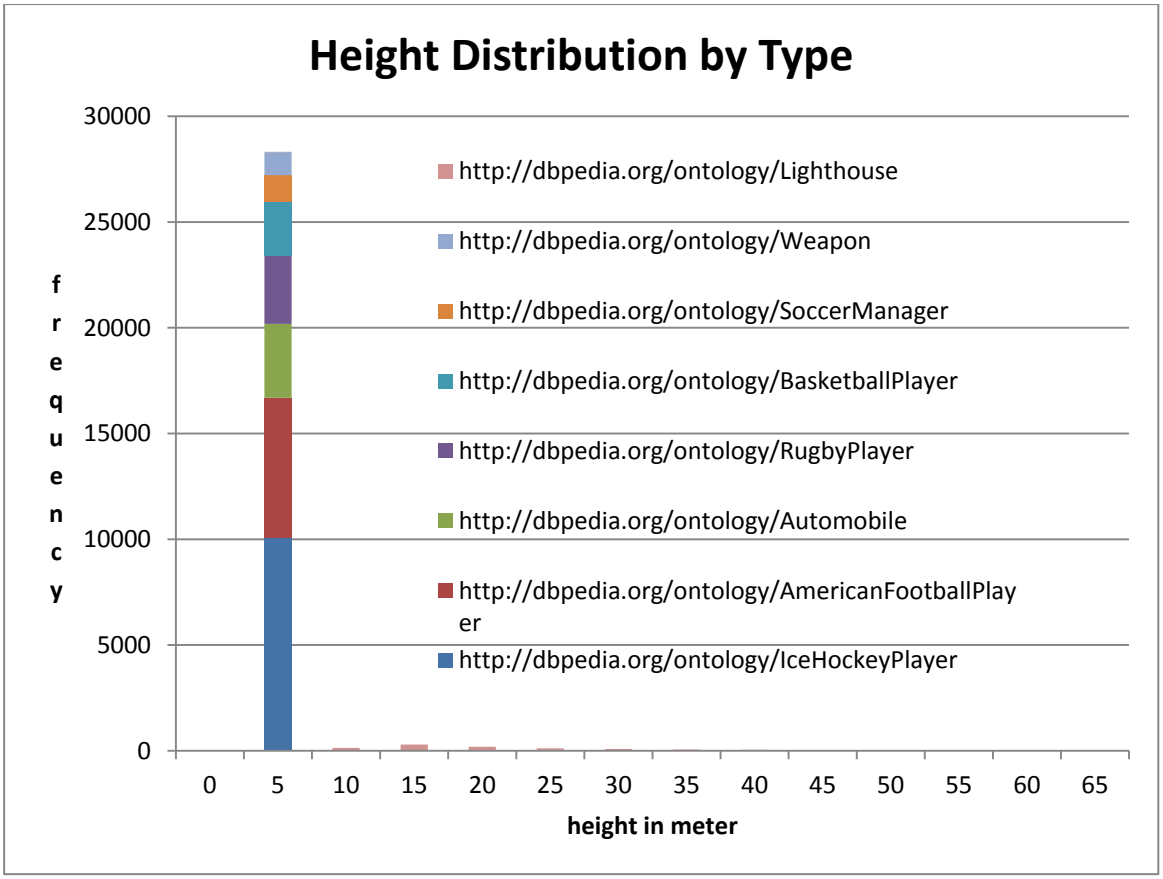

Figure 14: Height Distribution by Type

This chart gives the frequencies for persons only on an axis scaled to the usual heights of people. Now, we can clearly discern a normal distribution. Notable is the spike at 1.52 meters, which consists almost exclusively of incorrect values. Close inspection also shows that American football and basketball players are much taller than the average athlete.

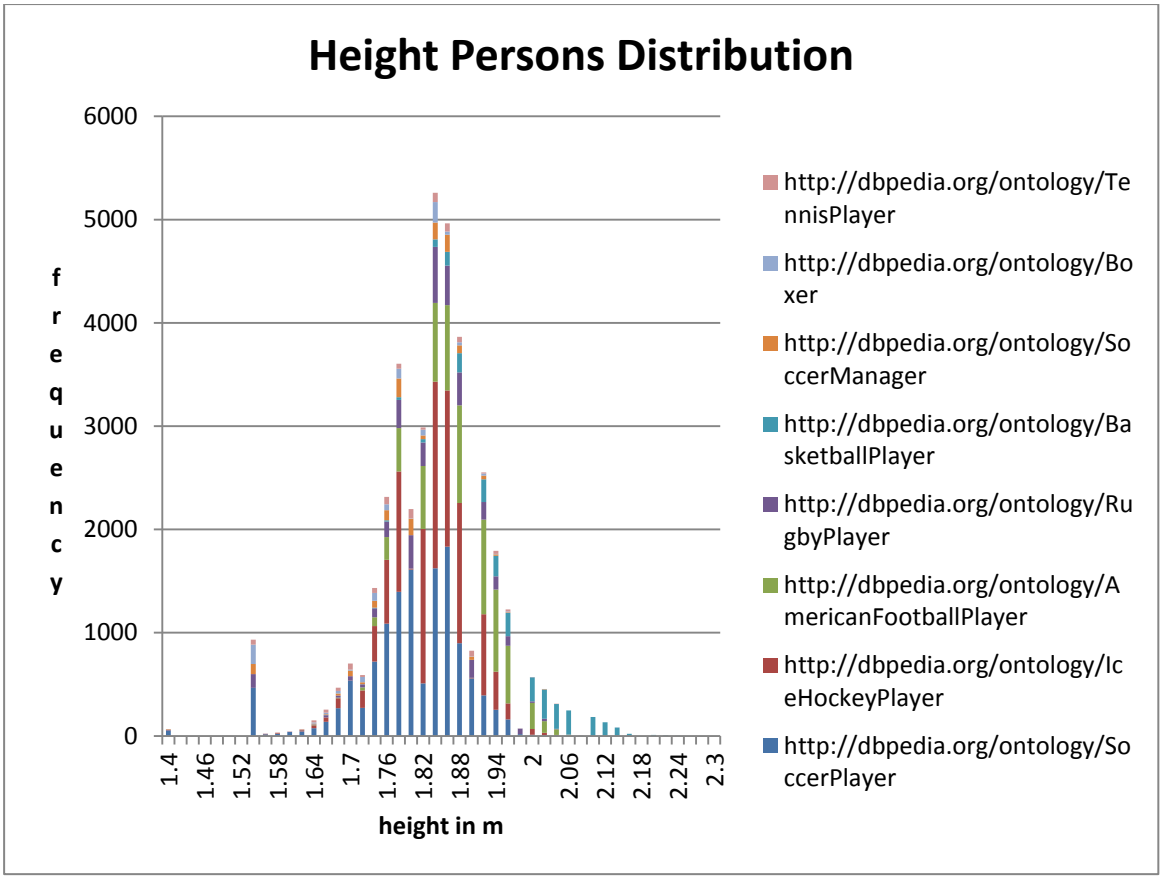

Figure 15: Height Persons Distribution

4.5.3 Elevation

The distribution for *elevation* is similar to that of *populationTotal* but also includes negative values (for example the Dead Sea) though its overall range is far more limited. There are 206,997 triples in total, all of which were successfully parsed. Valid values range from about -1000 for an undersea volcano to almost 9000 for the Mount Everest.

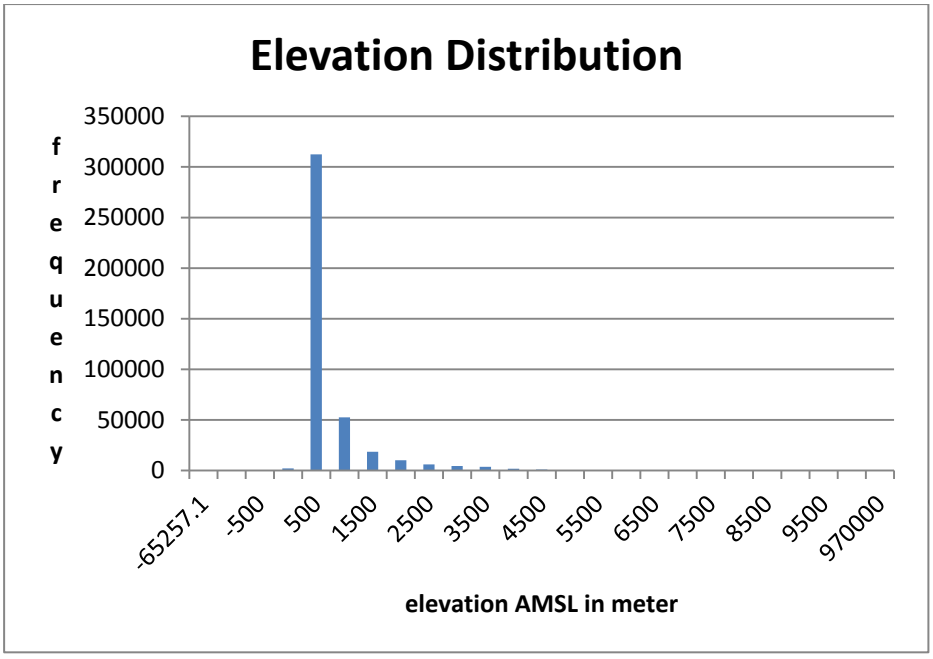

Figure 16: Elevation Distribution

Splitting the dataset by type does not reveal much new information. This is to be expected, considering that settlements make up most of the subjects and - unlike with populationTotal - there is no direct correlation between the elevation of a place and the kind of settlement it constitutes.

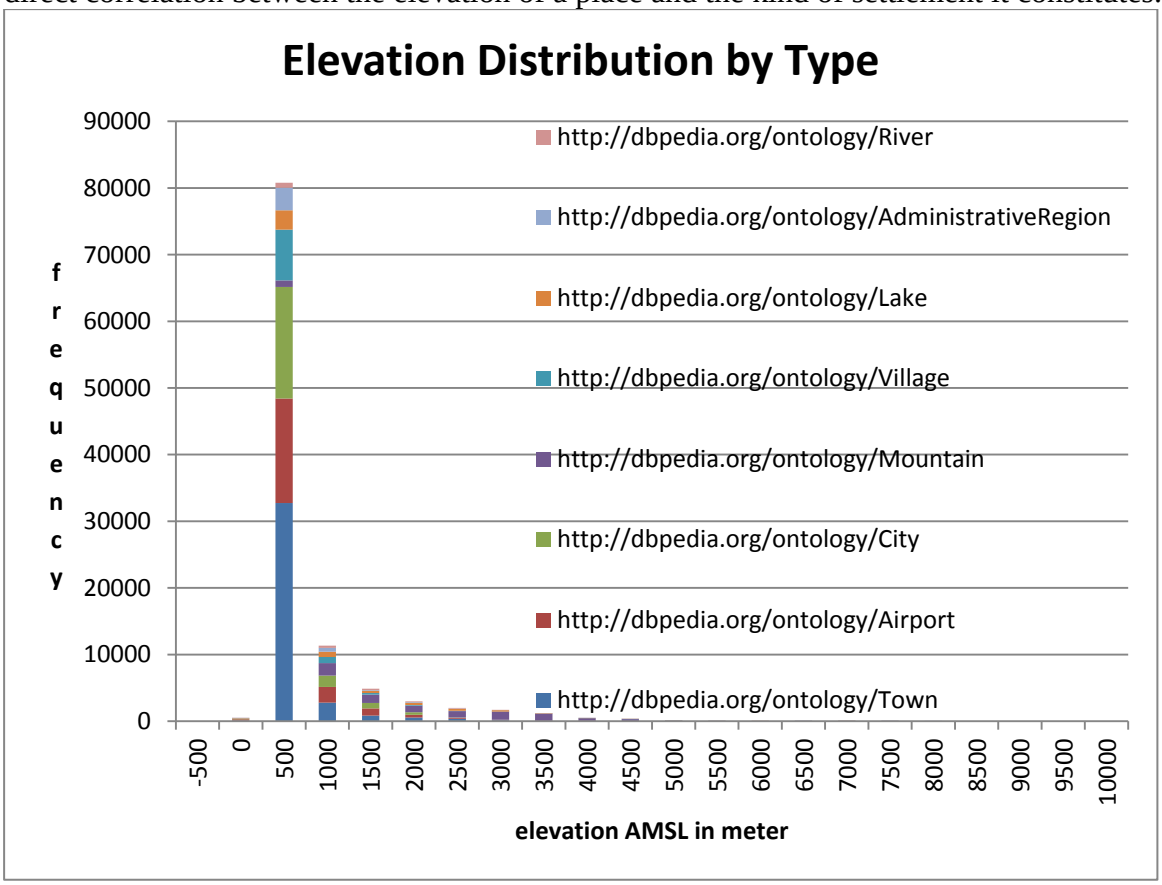

Figure 17: Elevation Distribution by Type

4.6 Comparisons

4.6.1 Mode Comparisons

We first determined reasonable values for the parameters responsible for the sensitivity of each method. The lower end of the sensitivity was always chosen to be the point beyond which only negligible change in the number of positives occurs. The upper limit of sensitivity was either chosen at the point where the absolute number of false positives becomes too high for the method to be actually useful (i.e. in the thousands) or at the point where the ratio of true to false positives became too low.
We evaluated the three datasets together to prevent optimization for a single predicate.
Again, we will take a look at the ROC curves first to understand the general performance of each method and mode. For these tests it was not possible to use a common axis scaling for all methods as the results differed by magnitudes from method to method.

This chart compares the different modes for IQR. The *default* mode performs clearly the worst, *byType* the best and *cluster* occupies the middle ground.

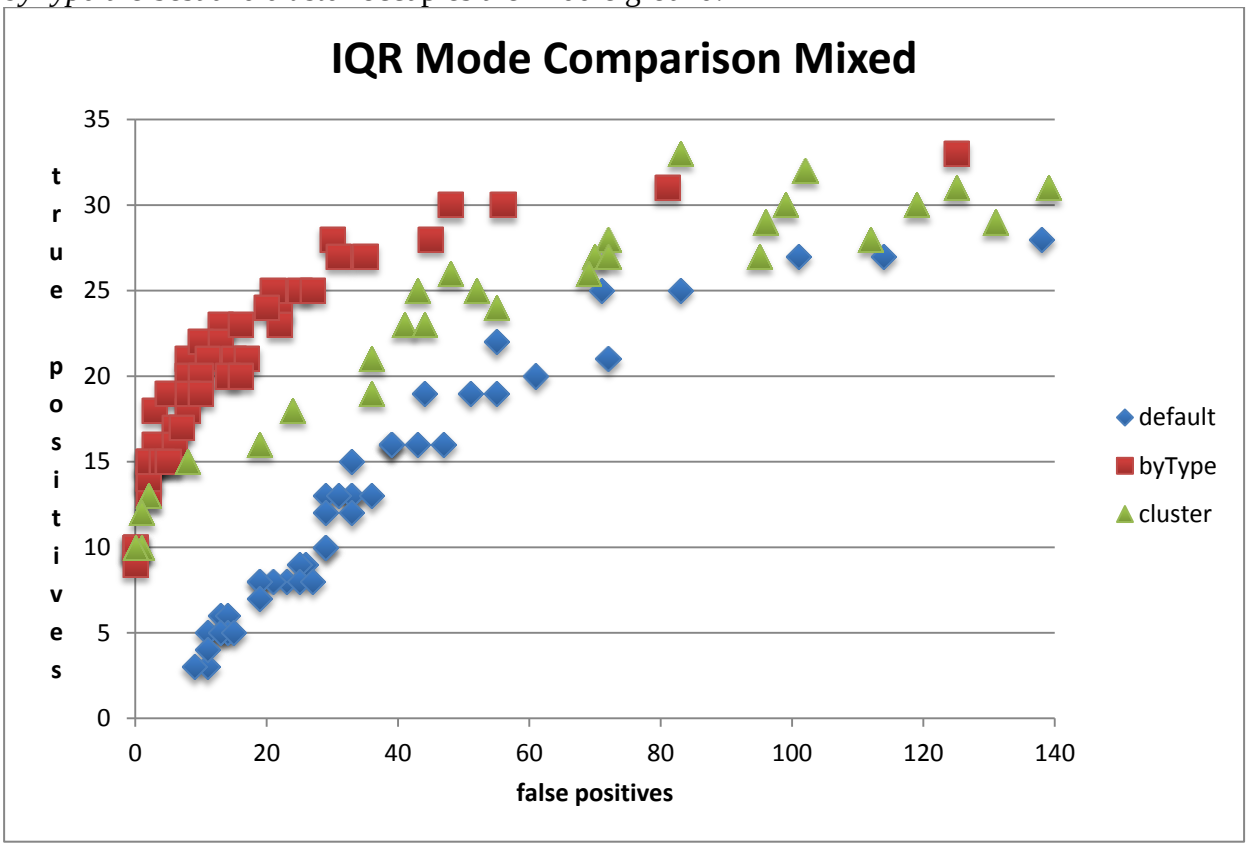

Figure 18: IQR mode comparison mixed

When we look at the same comparison for the *dispersion* methods, the results are somewhat surprisingly exactly reversed. While *cluster* starts out a little better than *default*, it later causes more false positive. *ByType* does not work at all with *dispersion* and yields by far the worst results.

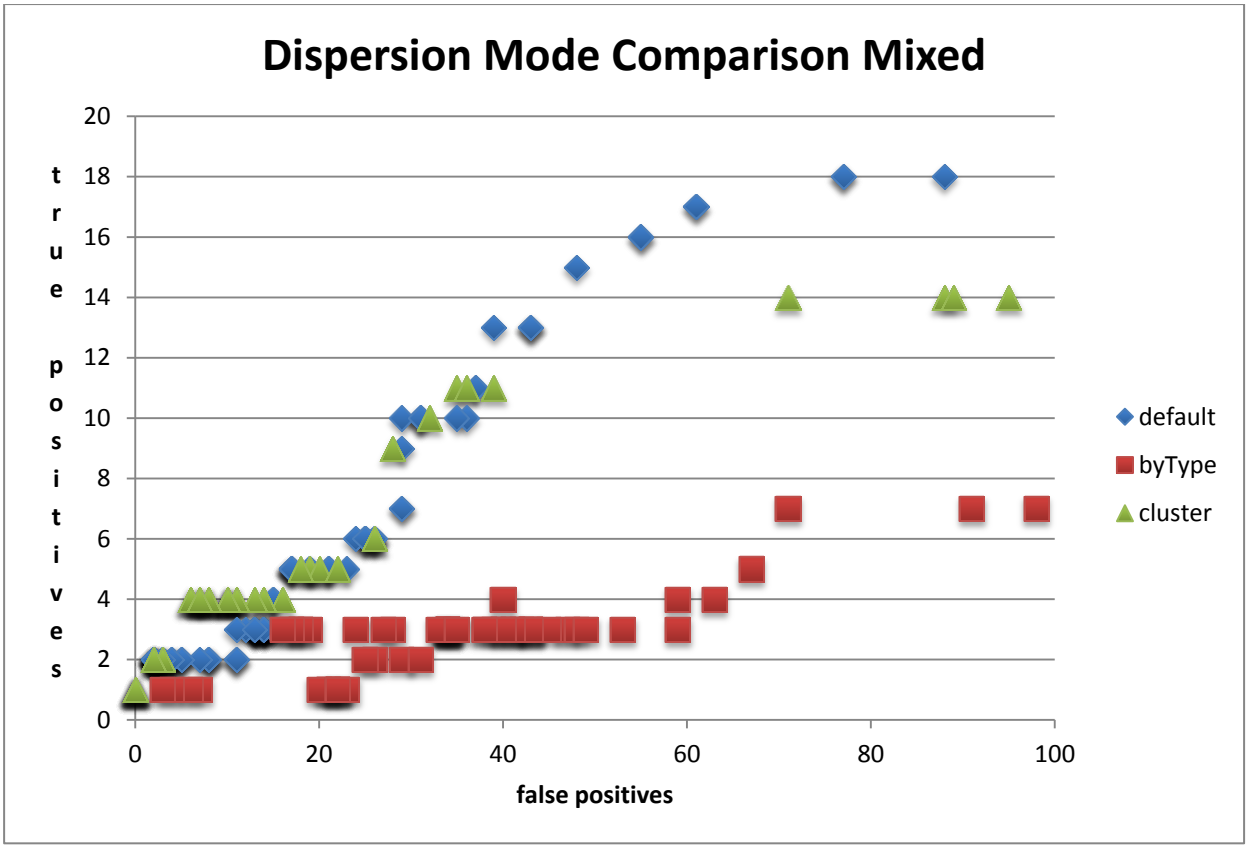

Figure 19: dispersion mode comparison mixed

With KDE-FFT, the cluster method shows by far the worst results, *byType* works slightly better than *default* but overall the true positive:false positive ratio does not seem to make this method usable.

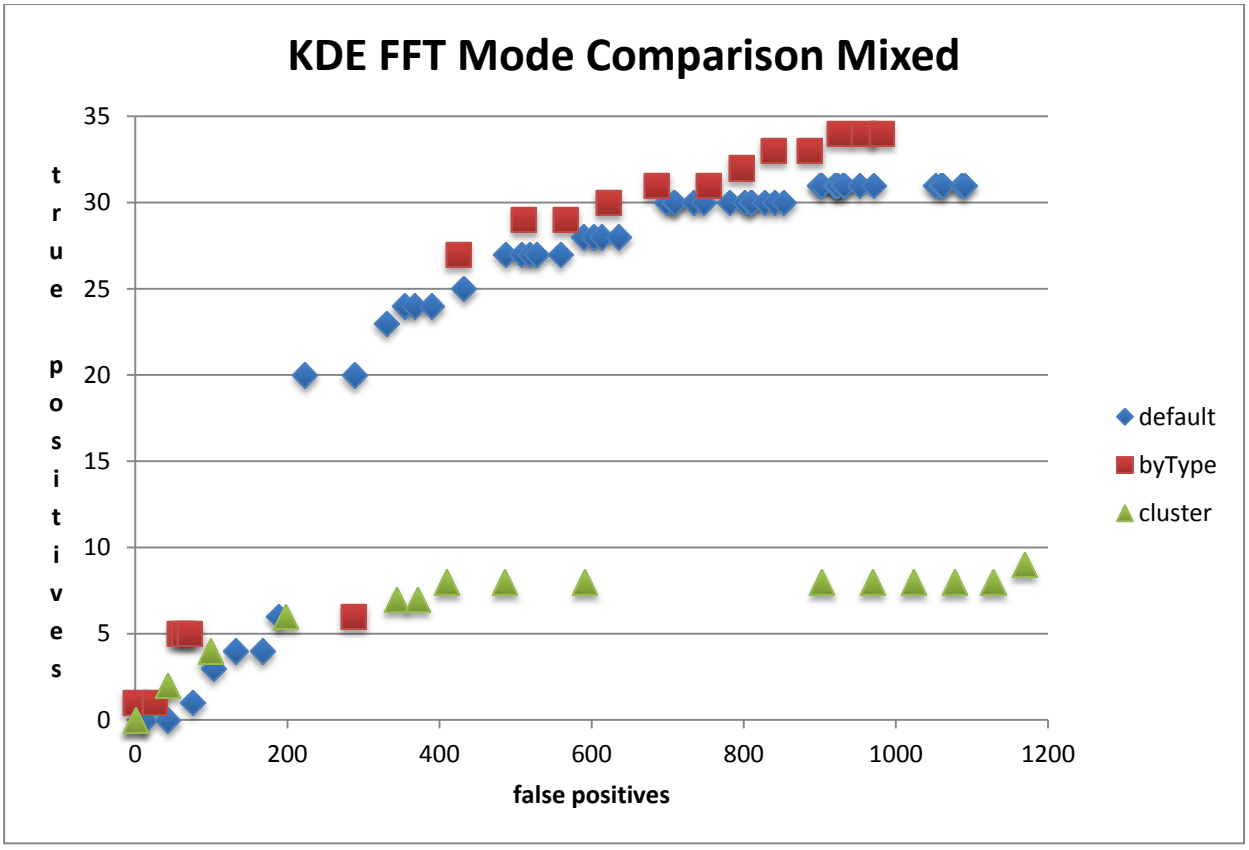

Figure 20: KDE-FFT mode comparison mixed

4.6.2 Dispersion Estimator Comparison

This chart gives a comparison of the four dispersion estimators. All four graphs overlap almost completely so in effect there does not seem to be much difference between the estimators.
First we have the comparison for the *default* case:

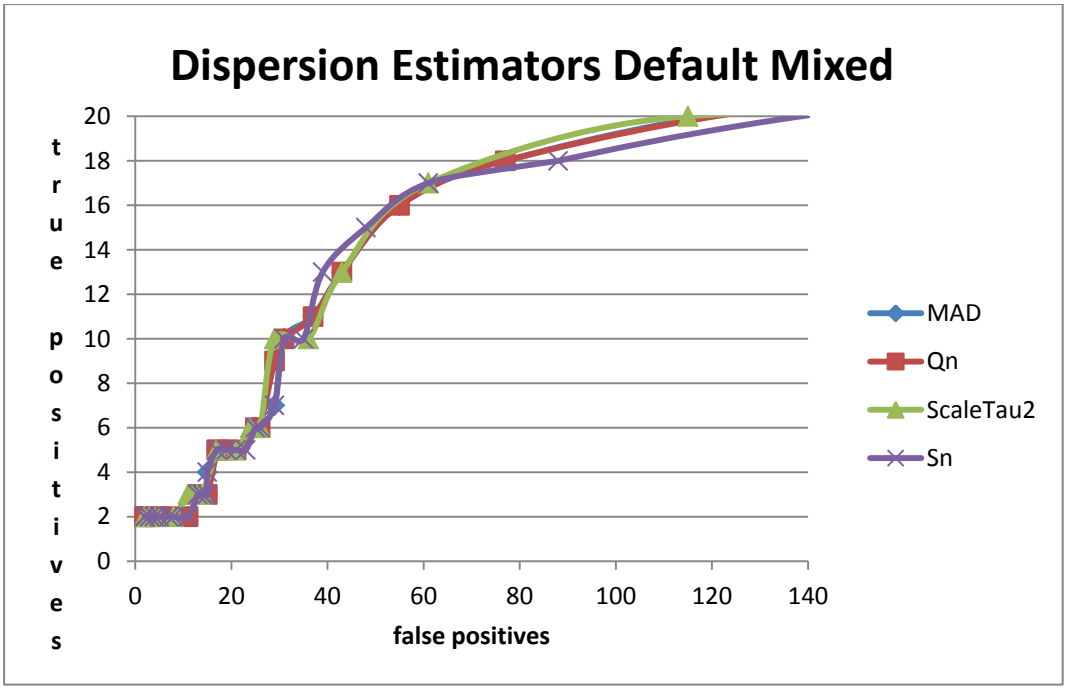

Figure 21: *dispersion* estimators *default* mixed

For the *byType* case ScaleTau2 falls slightly off while the other three are again almost identical.

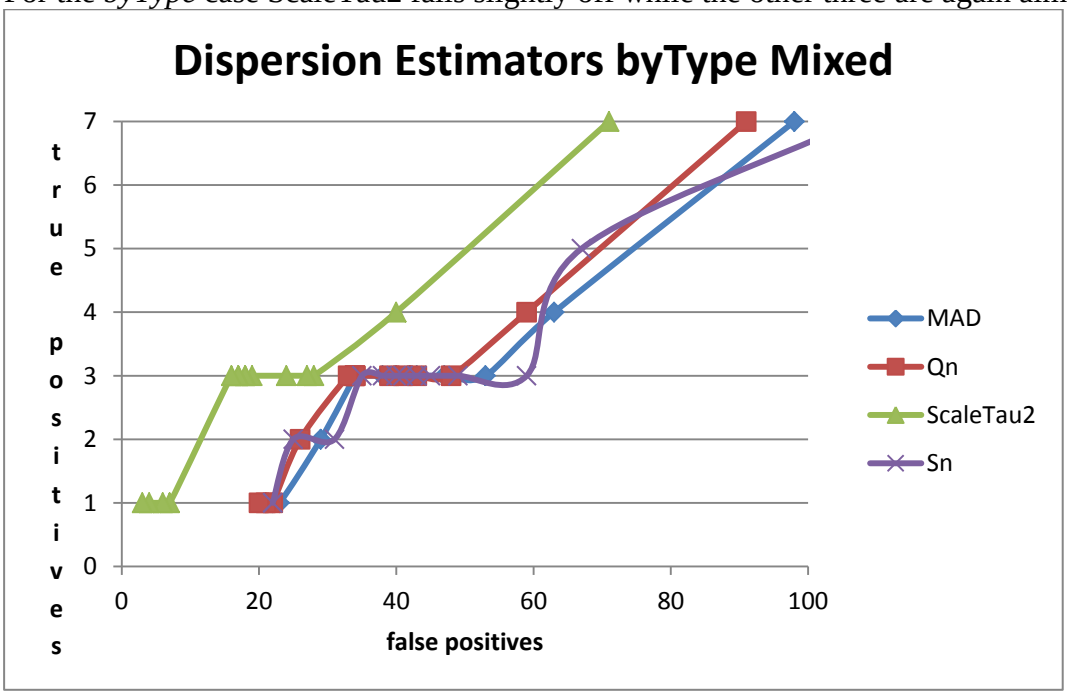

Figure 22: dispersion estimators *byType* mixed

And finally for the *cluster* case we do not see much difference either:

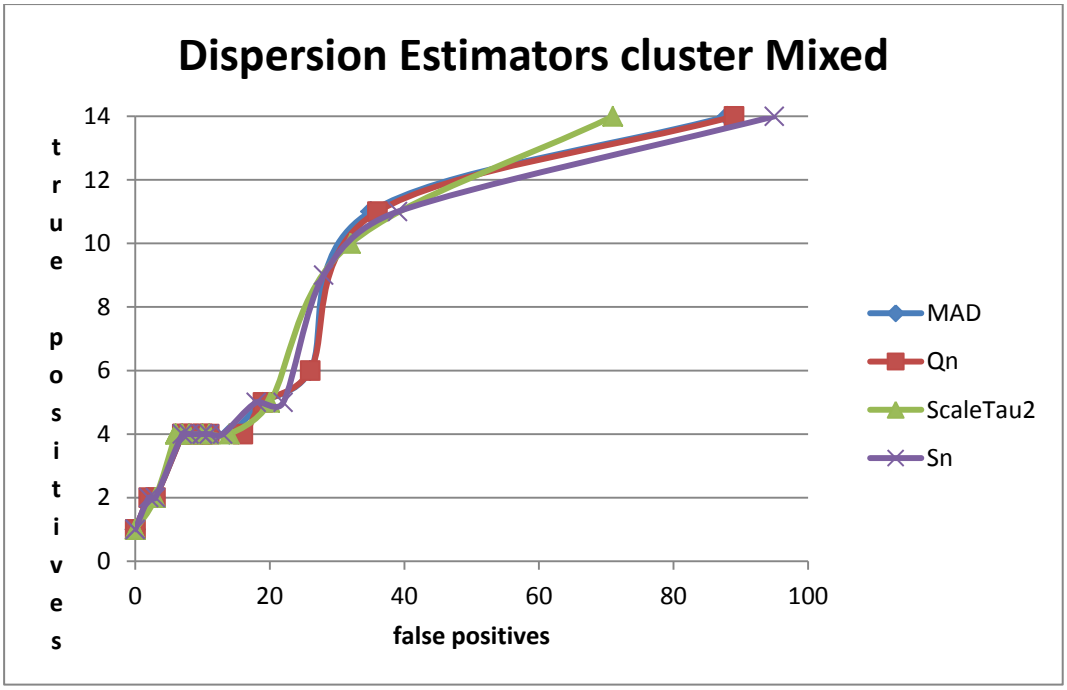

Figure 23: dispersion estimators cluster mixed

4.7 Iteration

The last mode that we have not evaluated yet is *iterative*. Since this mode does not affect the way the data is separated, it can easily be combined with the other three modes. We performed the tests on the same data and with the same parameters as in the last section (see appendix, "Iteration Results").

For the dispersion and IQR methods, we do not see much change; this is to be expected, as the methods used to estimate the dispersions are designed to be robust to outliers, so removing them should not have much effect.

For FFT-KDE, iterative application only amplifies the already bad performance for a single iteration.

However, we will see some interesting results with regards to iterations in the following KDE section.

4.8 Runtime

Runtimes for one analysis run in *default* mode on the three datasets were 1832ms for IQR, 2,297ms for *dispersion*, 6,922ms for KDE-FFT and 2,469,011ms (over 41 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 *byType* mode, IQR takes 41,538ms, KDE-FFT 31,336ms and dispersion 72,852 for one sample run.

The *cluster* mode suffered from extremely high runtimes of around one hour for the *height* dataset and over 24 hours for *elevation* and *populationTotal*.

4.9 KDE

The high runtimes of the exact KDE implementation prevented us from putting it through the same test tracks as the other methods.

But as KDE showed promising results on the simulated data and was successfully used for outlier detection in Linked Data by (Fleischhacker, 2013), we did not want to forego the chance of evaluating it all together.
We therefore chose to evaluate on the smallest of the three datasets, *height*, only.
Firstly, we have again the basic comparison of the three modes:
Default performs extremely bad but *byType* and especially *cluster* show promise with the best result being 51:9 true positives:false positives.

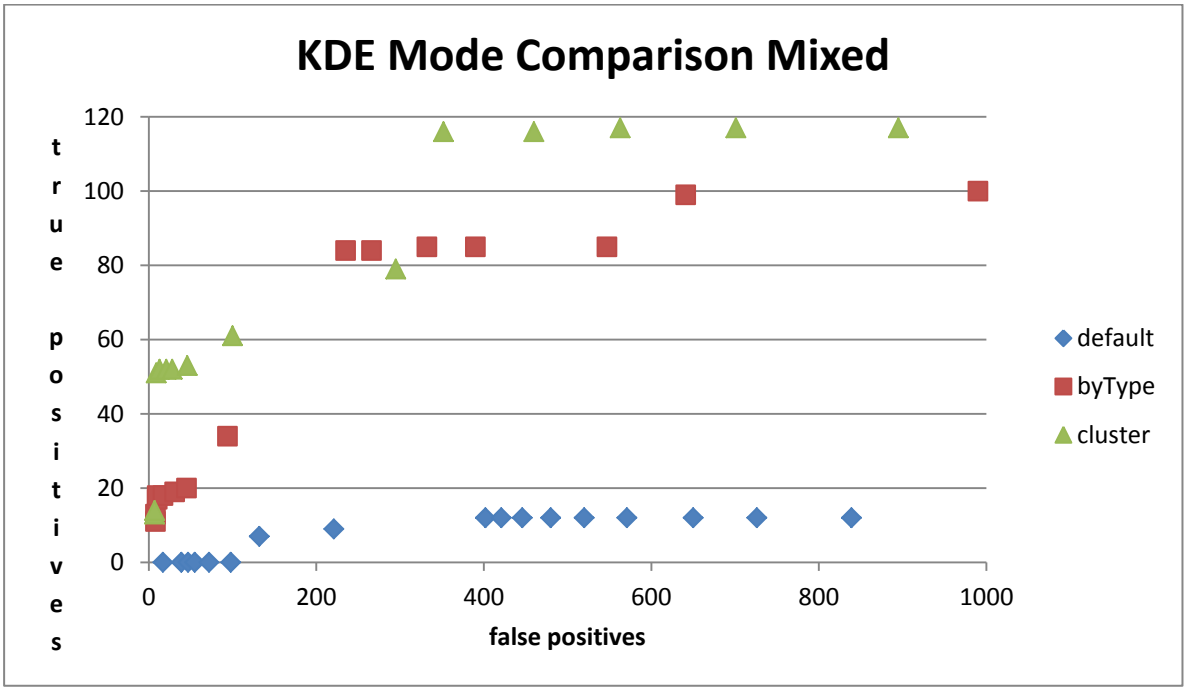

Figure 24: KDE mode comparison mixed

Things get even more interesting when we turn to iterative application of KDE because for the first time, we observe a significant improvement with increasing numbers of iterations.

At first, in *default* mode, we again observe the pattern of more iterations deteriorating already bad results.

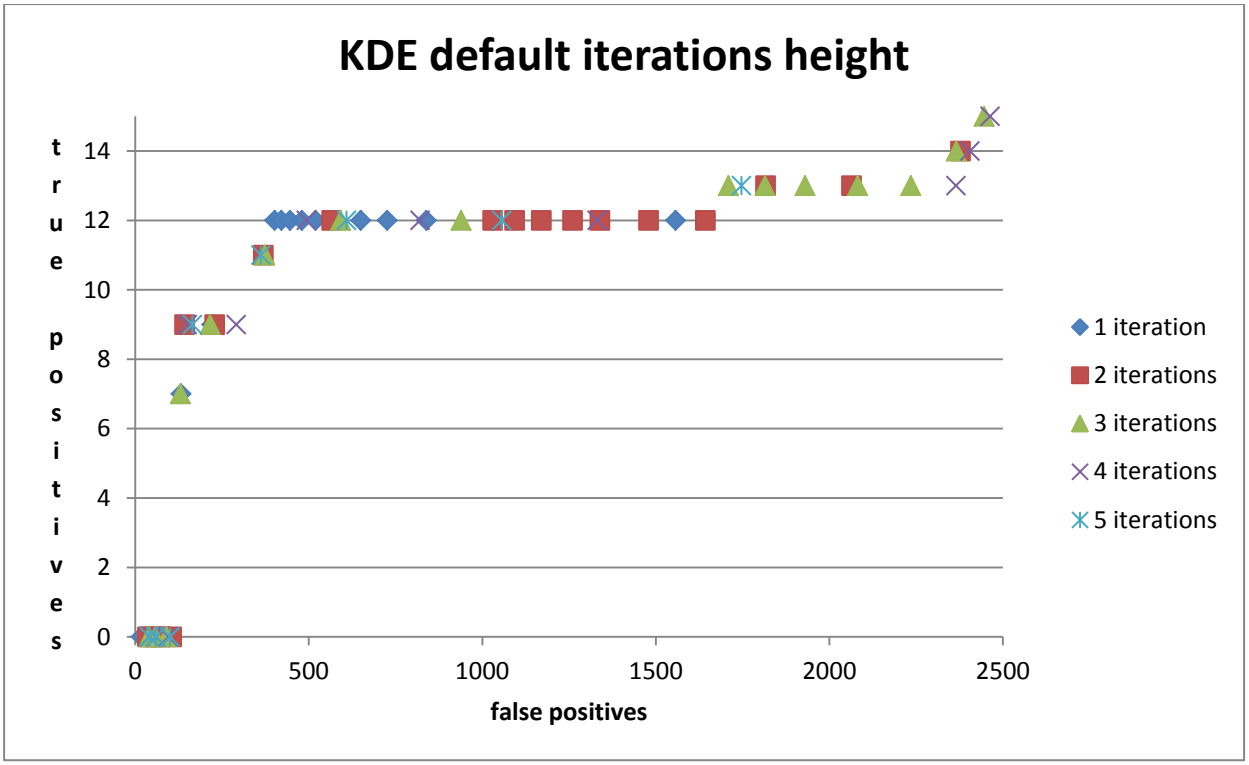

Figure 25: KDE *default* iterations height

However, once we apply multiple iterations to the already good results of *byType*, we see a significant improvement after the second iteration and some further improvement after the third. Beyond that, not much change is observed.

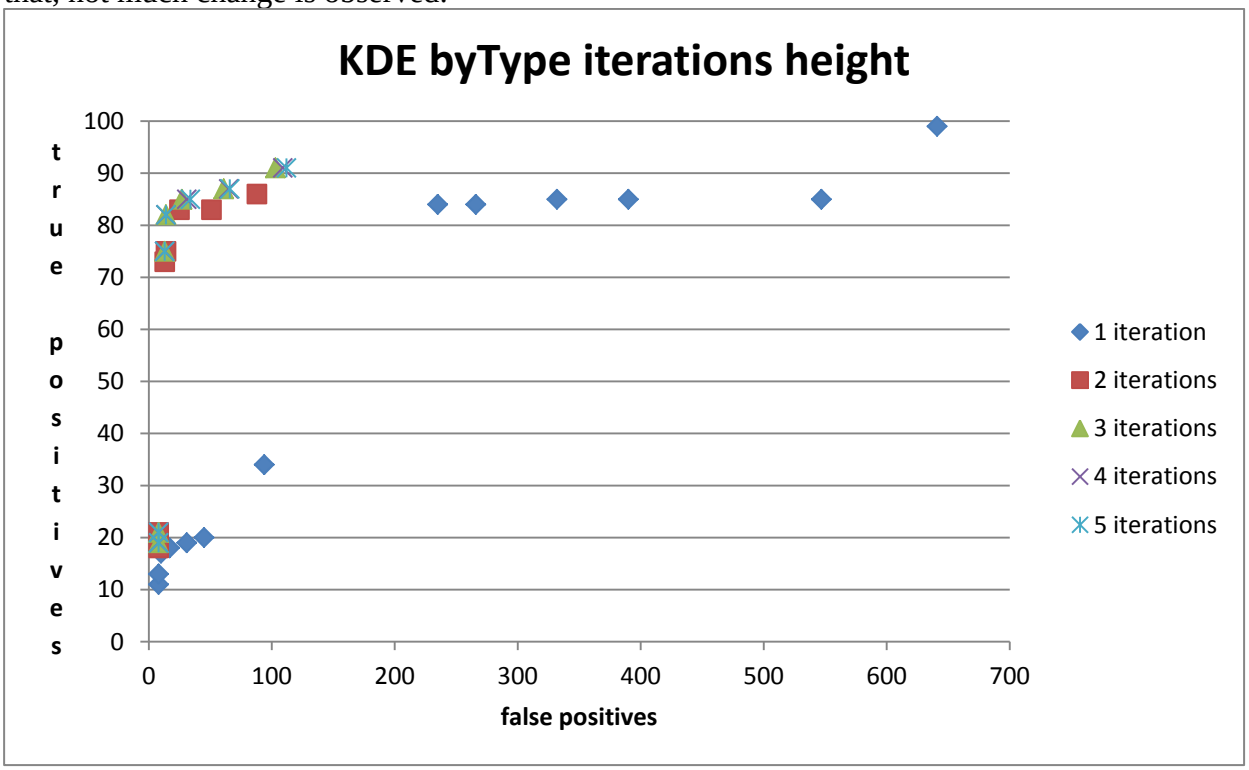

Figure 26: KDE *byType* iterations height

The chart for the cluster mode paints a similar picture, though overall results are not quite on the same level as for *byType*.

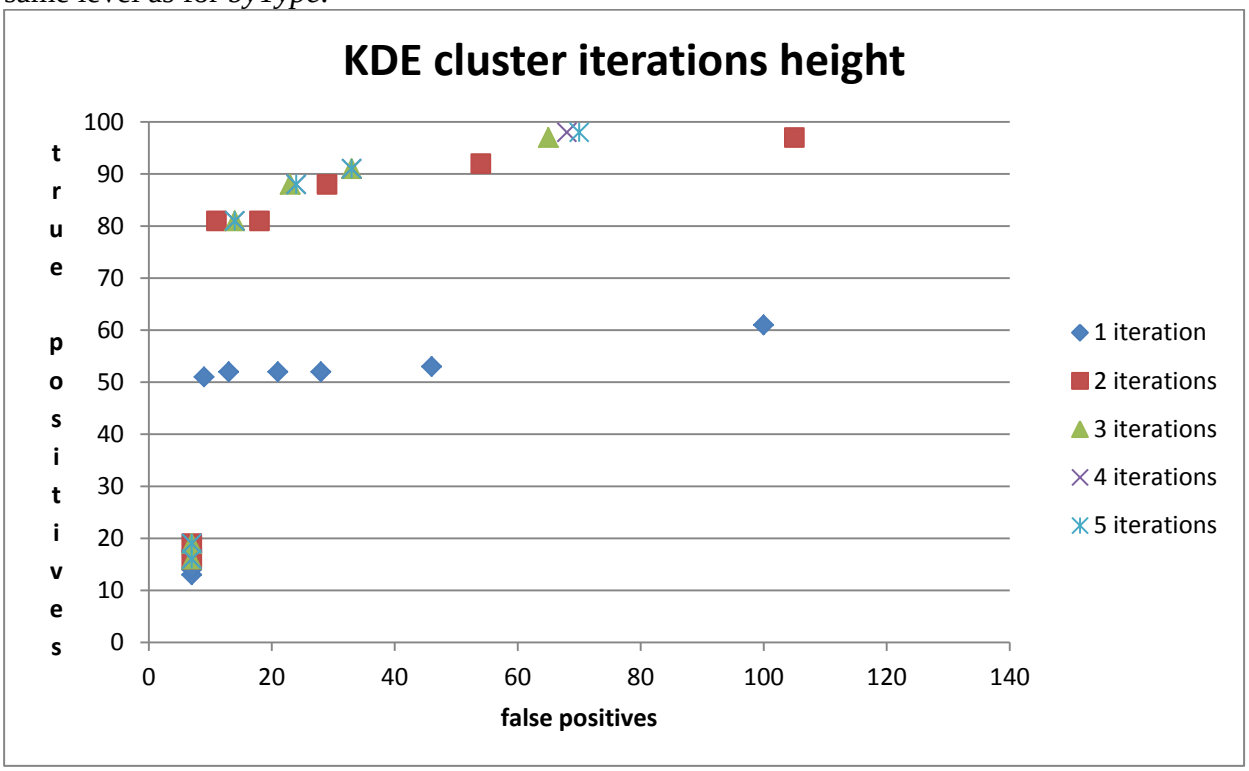

Figure 27: KDE cluster iterations height

The point at 82 true positives : 14 false positives (85% precision) is one of the best results overall, only equaled by IQR, albeit in much smaller numbers, 15:2 (88%). However, ultimately those results should be taken with a grain of salt, as they are only based on a single dataset, which makes the evaluation prone to over-optimization for this specific predicate.

Our interpretation of these results is that because, unlike the other methods that are based on inherently robust statistics, which are supposed to ignore outliers, KDE weighs all data points equally. Thus, the removal of outliers can actually affect the outlier score of other outliers that had previously been masked.

4.10 Parameters

These charts give a general idea of how the different algorithms and modes would be expected to perform. However, in order to actually utilize them we need to choose concrete values for the parameters.

4.10.1 Quality Measure

We recorded the number of detected outliers, the number of true positives, the number of false positives and the number of false negatives for each run. True positives (tp) are data points, which were detected as outliers and which are in fact incorrect. False positives (fp) are data points, which were detected as outliers but were actually correct. False negatives (fn) are incorrect values, which were not identified as suspicious by the outlier detection method.

From this we calculated the precision, $\frac{tp}{tp+fp}$, and the recall, $\frac{tp}{tp+fn}$. We chose not to use the frequently used F-measure, $2\,\frac{precision \cdot \text{recall}}{precision + recall}$, as the ultimate measure of quality since for our purposes precision is much more important than recall because of the great number of incorrect values that cannot be identified as such by outlier detection methods alone without causing a massive number of false positives. Precision on the other hand is important as the stated goal of this thesis is to find unsupervised methods for identifying incorrect values and a high number of false positives would effectively mean a higher need for human interaction.
Thus, we used $tp \cdot precision^k$, where $k = 2$ or 3, as a measure of an algorithm's quality to optimize for a high total count and high precision. We will call this measure the "precise total".

We will now take a look at the most successful methods and modes as judging by our previous examinations. That means all combinations that were able to produce more true than false positives for more than a few triples. Namely the combination of IQR and *byType* or *cluster* and the combination of KDE, *byType* or *cluster* and *iterative*.

This chart depicts the parameters and results that yielded the highest precise total measures.
Low quantile values perform well, which is to be expected if we do not assume the number of incorrect values to exceed the quantile. Multipliers for the IQR of 30-50 appear useful. With values of 10-20, we can already recognize an increase in the number of false positives.

4.10.2 IQR

truePositive	falsePositive	precision	preciseTotal2	preciseTotal3	multiplier	quantile
18	3	0.857142857	13.2244898	11.33527697	20	0.006
19	5	0.791666667	11.90798611	9.427155671	20	0.011
15	2	0.882352941	11.67820069	10.30429473	30	0.006
15	2	0.882352941	11.67820069	10.30429473	50	0.011
16	3	0.842105263	11.34626039	9.55474559	30	0.011
21	8	0.724137931	11.01189061	7.974127681	10	0.006
14	2	0.875	10.71875	9.37890625	40	0.006
14	2	0.875	10.71875	9.37890625	50	0.006
15	3	0.833333333	10.41666667	8.680555556	40	0.011

Table 1: IQR *byType* parameters

The combination of IQR and *cluster* proved quite successful too. Useful multipliers are again between 20 and 50, while the efficient quantile values are 0.005 smaller on average.

truePositive	falsePositive	precision	preciseTotal2	preciseTotal3	multiplier	quantile
12	1	0.923076923	10.22485207	9.438324989	30	0.001
10	0	1	10	10	50	0.001
13	2	0.866666667	9.764444444	8.462518519	20	0.001
10	1	0.909090909	8.26446281	7.513148009	40	0.001
15	8	0.652173913	6.379962193	4.160844908	10	0.001
25	43	0.367647059	3.379108997	1.242319484	20	0.006
16	19	0.457142857	3.343673469	1.528536443	50	0.006
18	24	0.428571429	3.306122449	1.416909621	40	0.006
26	48	0.351351351	3.209642075	1.12771208	30	0.011

Table 2: IQR *cluster* parameters

4.10.3 KDE

Next, we take a look at the results of the basic KDE implementation being applied to the *height* dataset.

The parameters for both *byType* and *cluster* mode look very similar. More than two iterations do not improve results in this case and a threshold of 0.3 or 0.4 seems most useful. With a threshold of 0.93 for *byType* or 0.92, we are able to uncover a vast number of outliers (the people with an erroneous height of 1.524m) at the expense of a large number of false positives. However, we would argue that this success is very specific to this dataset and in general such high thresholds should be avoided.

truePositive	falsePositive	precision	preciseTotal2	preciseTotal3	maxIterations	threshold
956	1292	0.425267	172.8945	73.52629	3	0.93
82	14	0.854167	59.82726	51.10245	3	0.4
82	14	0.854167	59.82726	51.10245	4	0.4
82	14	0.854167	59.82726	51.10245	5	0.4
75	13	0.852273	54.47766	46.42982	3	0.3
75	13	0.852273	54.47766	46.42982	4	0.3
75	13	0.852273	54.47766	46.42982	5	0.3
75	14	0.842697	53.26032	44.88229	2	0.4
73	13	0.848837	52.5983	44.64739	2	0.3

Table 3: KDE *byType* parameters

truePositive	falsePositive	precision	preciseTotal2	preciseTotal3	maxIterations	threshold
81	11	0.880435	62.7884	55.28109	2	0.3
81	14	0.852632	58.88543	50.20758	3	0.3
81	14	0.852632	58.88543	50.20758	4	0.3
81	14	0.852632	58.88543	50.20758	5	0.3
81	18	0.818182	54.22314	44.36439	2	0.4
88	23	0.792793	55.3098	43.84921	3	0.4
667	996	0.401082	107.2983	43.03547	2	0.92
88	24	0.785714	54.32653	42.68513	4	0.4

Table 4: KDE cluster parameters

4.11 Random Sample

In order to evaluate on a representative combination of datasets, we randomly selected 50 resources using the SPARQL query `SELECT distinct ?x WHERE {?x a owl:Thing} ORDER BY RAND() LIMIT 50 OFFSET 1234`. For each resource we then read the data of all predicates. During parsing we filtered out datasets with less than 100 data points to allow for significant statistical results. Furthermore, we only used datasets where more than 50% of all data points could be successfully parsed as lower rates indicate that the data is in fact not numerical.

Data Acquisition

One problem with this evaluation approach is that DBpedia's SPARQL endpoint is not geared for mass downloading data and places various restrictions on requests. One being that a single query can only return 40,000 triples. For queries with larger result sets we have to order the result set and then walk it using an offset.

Beyond this limit on the absolute number of triples returned, there seems to be a bandwidth limit that applies when the result of a single query exceeds a certain number of bytes. That is, for a predicate with numerical objects 40,000 triples will generally return fine but when dealing with a predicate such as `dbpedia-owl:abstract`, which contains a description of a few sentences for each subject in multiple languages, this limit will be hit. To mitigate this we can iteratively reduce the window size but this decreases performance even further to the point where acquiring all data for `dbpedia-owl:abstract` would take days. Since this problem occurs only with predicates that have long strings as objects – exactly the kind of data that we are not interested in for numerical analysis – we decided to exclude those predicates by hand in order to prevent automatic queries from getting stuck. This resulted in a list of 15 excluded predicates:
`http://www.w3.org/2000/01/rdf-schema#comment`,
`http://dbpedia.org/ontology/thumbnail`,
`http://dbpedia.org/property/wikiPageUsesTemplate`,
`http://dbpedia.org/property/hasPhotoCollection`,
`http://purl.org/dc/terms/subject`,
`http://dbpedia.org/ontology/wikiPageExternalLink`,
`http://www.w3.org/2000/01/rdf-schema#label`,
`http://www.w3.org/2002/07/owl#sameAs`,
`http://www.w3.org/ns/prov#wasDerivedFrom`,
`http://xmlns.com/foaf/0.1/depiction`,
`http://xmlns.com/foaf/0.1/isPrimaryTopicOf`,
`http://dbpedia.org/ontology/abstract`,
`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`,
`http://dbpedia.org/property/name` and `http://xmlns.com/foaf/0.1/name`.
None of those contain numerical data and they can therefore be excluded from analysis. Other than that resources/predicates were chosen completely at random.
After parsing, this left us with 13,053,007 triples in 168 datasets to work with.

4.12 Evaluation on 50 Random Resources

The combination of IQR and *byType* had shown high precision (88%) as well as runtimes that seemed suitable for large scale analysis, so we chose to evaluate this combination further on the representative sample of 50 random resources we collected.
For a first test, we used the parameters that had shown the highest precision (Quantile=0.011, multiplier = 50) to keep the need for manual analysis as low as possible. This yielded 1,703 suspicious triples, which we then evaluated by hand. 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 resources of the form "[some place],_[US state/borough]" (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 `http://dbpedia.org/property/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 discounting unknown values, IQR achieved a precision of 81% on this sample of 50 random resources.
Based on this successful run, we created the full ROC plots for IQR on the 50 resources.
We again used parameters for high precision only to allow for semi-manual analysis of all outliers.

The highest precision is achieved at 88% with 859 true positives and 108 false positives.

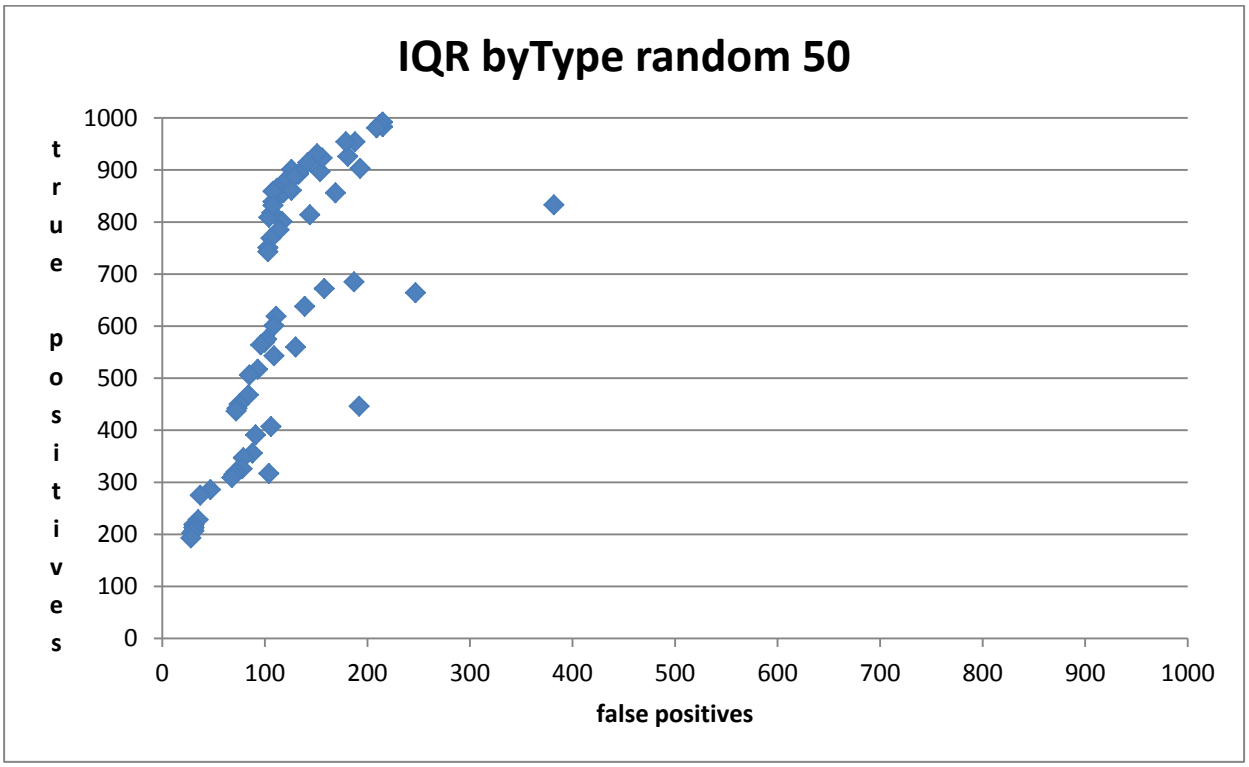

Figure 28: IQR *byType* random 50

The parameters for the most useful runs can be seen here:

falsePositive	truePositive	precision	preciseTotal2	preciseTotal3	multiplier	quantile
302	1330	0.81495098	883.3129836	719.856782	30	0.009
414	1421	0.774386921	852.1383219	659.8847714	20	0.009
126	901	0.877312561	693.4792738	608.3980776	50	0.007
233	1038	0.81667978	692.3105654	565.39604	50	0.009
151	931	0.860443623	689.2781655	593.0850019	50	0.008
142	914	0.865530303	684.7164328	592.6428216	60	0.008
121	881	0.879241517	681.0708334	598.8257527	60	0.007
112	865	0.885363357	678.0460573	600.3171336	70	0.007
108	859	0.888314374	677.8389854	602.1341142	80	0.007

Table 5: IQR *byType* random 50 parameters

We 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 *byType* mode, its triple has to have a useful type. However, three predicates, `http://dbpedia.org/property/date`, `http://dbpedia.org/property/years` and `http://dbpedia.org/property/postalCode` were responsible for the vast majority of those outliers. For those predicates, identifying true positives is extremely easy because years and dates with more than four digits do not make sense and the same goes for post-codes with more than ten digits.

By merely counting the corresponding objects for those three predicates we would achieve true positive:unknown/ false positive ratios of 7838:8751 (89%) with *dispersion* (MAD, constant factor = 300,000) and 5917:7216 (82%) with IQR (quantile = 0.01, multiplier = 41).
To verify that the results of our first test run were not dependent on only a few low quality datasets, we chose to evaluate on the higher quality *dbpedia-OWL* properties only as well, which left us with 4,150,26738 triples (32.8%) in 38 datasets (22.6 %) to analyze.
Since we are dealing with a subset, the number of true and false positives cannot increase. For the same sensitivities, we find fewer outliers but the overall excellent trend remains.

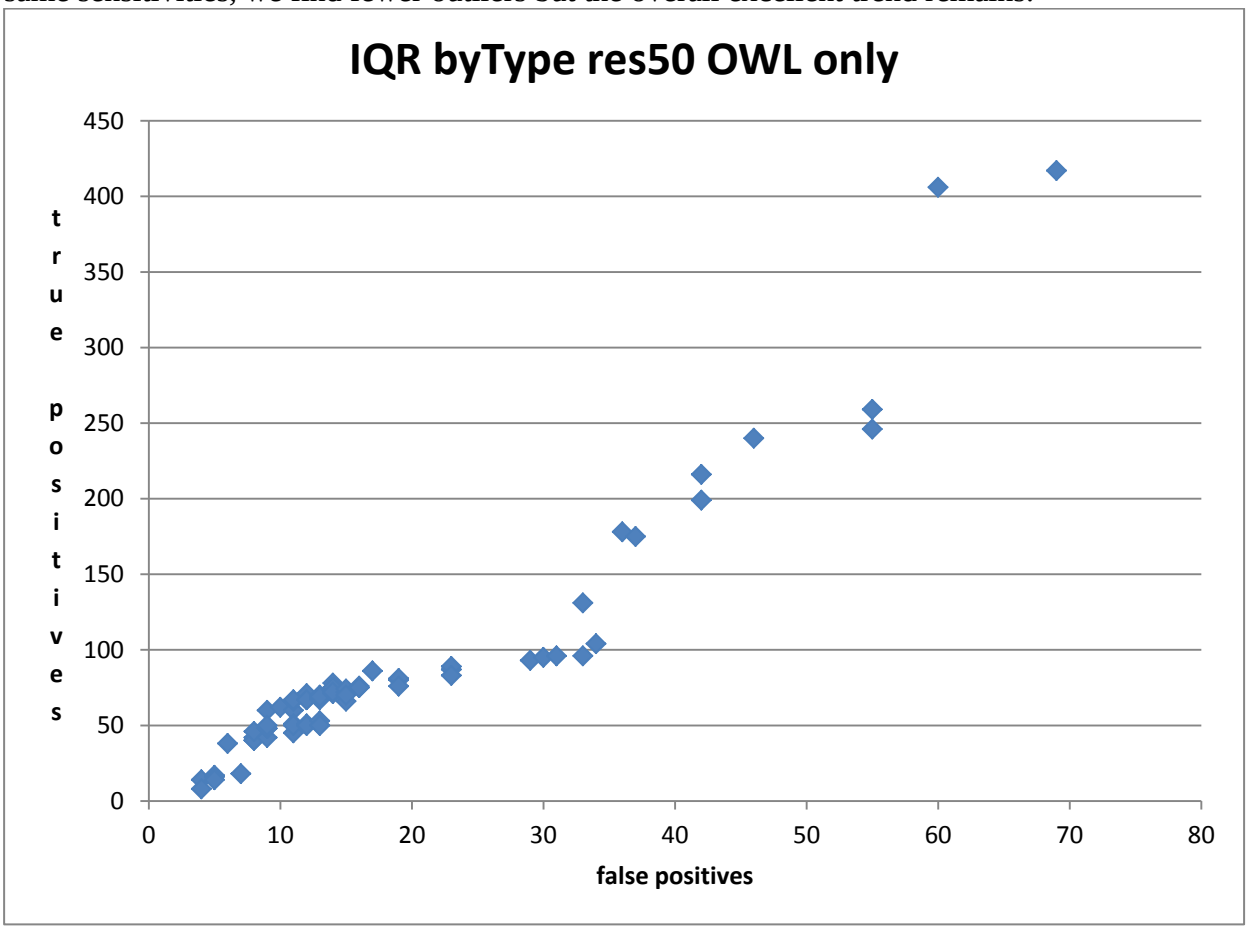

Figure 29: IQR *byType* random 50 OWL only

The parameters for the most useful runs can be seen here. We attribute the lower multiplier values to the smaller sample size. That is, with less overall datasets there may be less need for safety margins against certain datasets that are likely to cause large numbers of false positives.

falsePositive	truePositive	precision	preciseTotal2	preciseTotal3	multiplier	quantile
60	406	0.871245	308.1813	268.5013	10	0.007
69	417	0.858025	306.9981	263.4119	10	0.008
154	426	0.734483	229.8121	168.793	10	0.009
55	259	0.824841	176.2138	145.3484	10	0.006
46	240	0.839161	169.0058	141.8231	10	0.004
55	246	0.817276	164.3132	134.2892	10	0.005
42	216	0.837209	151.3986	126.7523	20	0.008
42	199	0.825726	135.6829	112.0369	20	0.007
36	178	0.831776	123.1494	102.4327	20	0.006

Table 6: IQR *byType* random 50 OWL only parameters

With precisions of over 80% for hundreds of outliers, the combination of IQR and *byType* has proven itself to be extremely useful. However, the parameters needed to achieve those precisions are still dependent on the respective dataset.

4.13 Data types

Methods based on data types lend themselves so evaluation on larger number of resources too.

4.14 Range

We first looked for inconsistencies between the declared range of a property and the actual data types of its objects. When a property has a declared range, all its objects should share the data type of that range. We found this to be the case for almost all predicates we inspected. The only exception are: `http://dbpedia.org/ontology/deathDate` and `http://dbpedia.org/ontology/birthDate` have a range of xsd:date but 886 (0.35%) and 1431 (0.23 %) objects respectively are instead of the data type `http://www.w3.org/2001/XMLSchema#gMonthDay`, thus featuring only a month and day but no year. Other than that, the only discrepancy occurred regarding properties with a range of xsd:nonNegativeInteger. The objects of those properties always were of type xsd:int exclusively but as all values were in fact non-negative, this should not cause much problems.

4.15 Data type Majority

Since range proved to be very consistent, we turned to analyzing the actual rates at which certain data types occurred for a given property as described in 3.1 Data Types.
We used a threshold of 80% for the main data type, so if more than 80% of the objects are of a single data type, we assume that data type to be the true data type for this property and every other one to be erroneous. The results for the random50 sample can be seen here:

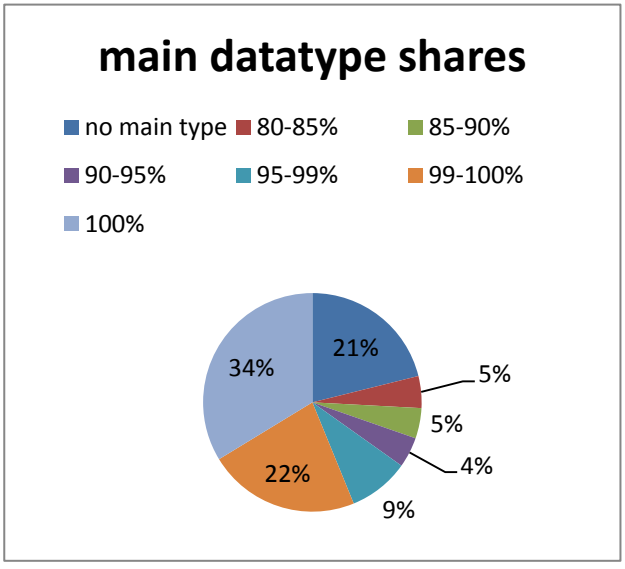

Figure 30: Majority Data type Shares

For about one in five resources, there is no data type that makes up at least 80% of the values. For about one in three, all objects are exclusively of one type.
For the remaining 45% we could easily detect objects

that have a wrong data type. Because frequently only the data type is wrong but the data itself is mostly correct, this can be used to identify triples that need parsing.

4.16 Parsing

In order to make the raw triples available for processing, we need to parse them first. Since the number format is standardized, the parsing phase can be used for error detection. If we assume that a certain property has numerical objects then all of its objects should be parsed successfully; otherwise we are dealing with erroneous data.

By looking at the distribution of the ratios of successfully parsed objects, we can identify two peaks. The first peak at less than 5% successfully parsed objects are the properties that do not have numerical objects and the same is likely true for the next few percentages. On the other side of the spectrum, we see the properties with 100% successfully parsed objects, where there is nothing to be done either. However, the second largest peak can be found right next to the 100% peak, in the 95% to100-ε% range. If over 95% of the objects of a certain property could successfully be parsed as numbers, this certainly indicates that the remaining 0.ε%-5% should be numeric too and thus require correction.

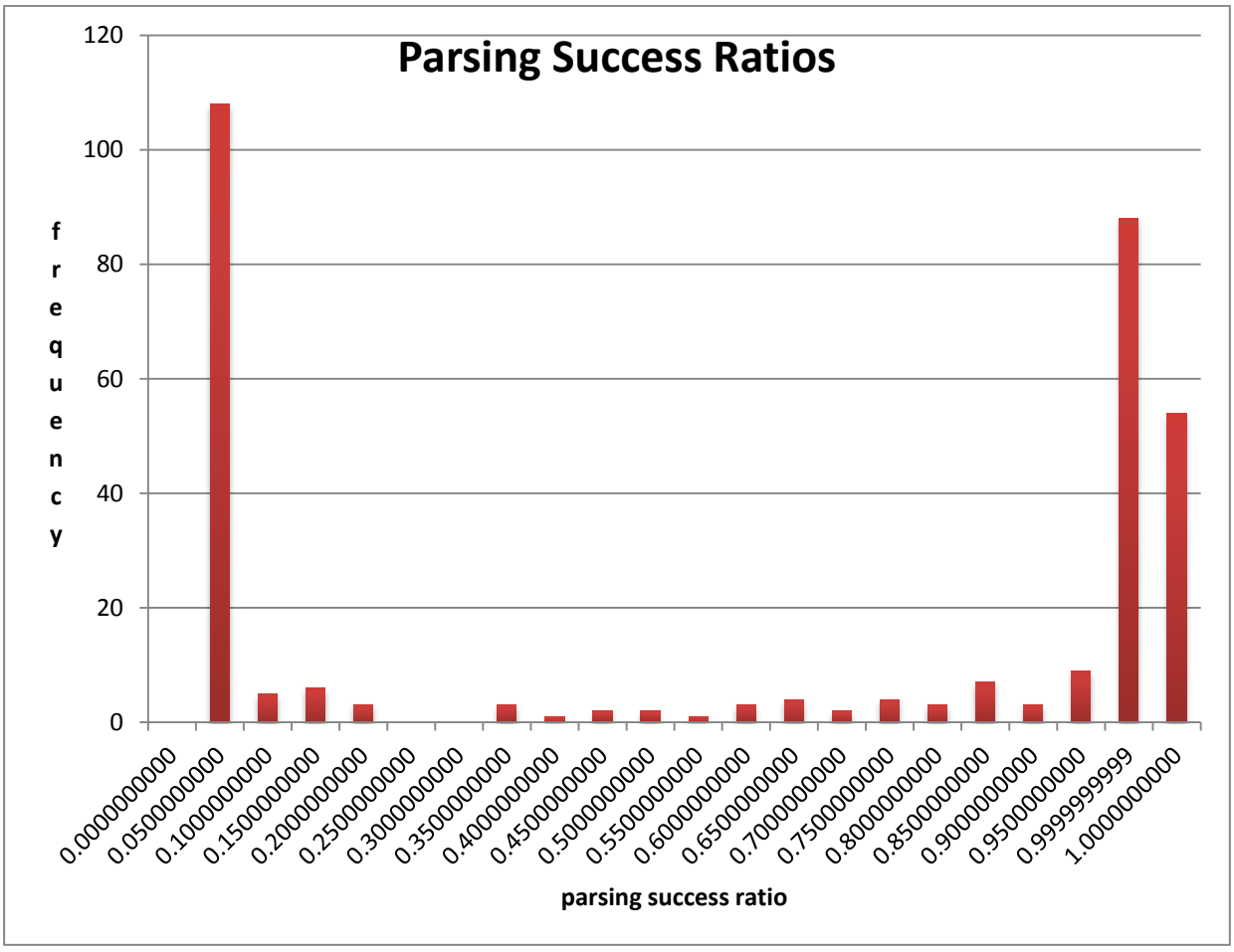

Figure 31: parsing success ratios distribution

5. DBpedia Analysis

We examined common patterns in the outliers we found to identify their causes. There are two basic classes of errors: those that exist already in Wikipedia and those that occur while parsing the data from Wikipedia to DBpedia. Wikipedia itself automatically imports data from sources like public population statistics, which makes it, like DBpedia, prone to parsing errors. On the other hand, user generated and manually entered data can suffer from inconsistencies, which can cause errors while parsing the data into DBpedia.

We now present a list of the error types that we identified. First for each of the three specific predicates we inspected and then general errors in DBpedia.

5.1 Errors in Wikipedia

In some cases, the data is corrupted at the source, i.e. the Wikipedia infobox. For example `http://en.wikipedia.org/wiki/Lerma,_State_of_Mexico` gives the elevation of a town in Mexico as "25,700 m 84,300 ft)". These errors are hard to quantify, as they do not seem to follow a specific pattern.

5.2 Dbpedia-owl:height

We will first look at the *height* dataset. This dataset contains 52,211 triples and we identified a total of 1,621 incorrect values. The distribution of the error sources we identified for those triples can be seen here.

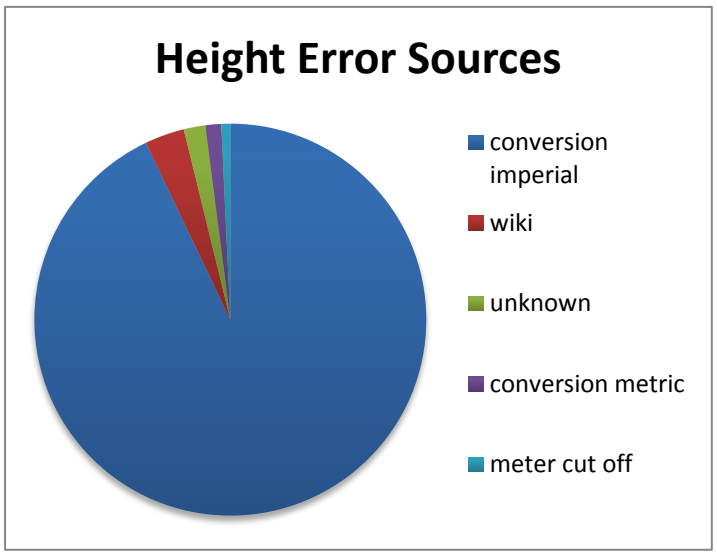

Figure 32: Height Error Sources

We will now explain those error sources in detail.

5.2.1 Imperial Conversion

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 (`http://dbpedia.org/resource/Ray_Wood`) is 1.80 meters in height according to Wikipedia but DBpedia gives his height as 1.524 meters. Another example would be locomotives with an incorrect height of 3.9624 meters or 13 feet, such as the "South_African_Class_NG10_4-6-2", (`http:// dbpedia.org/resource/South_African_Class_NG10_4-6-2`). In each case the correct value would only add a few inches or less than 0.3 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.

5.2.2 Metric Conversion

In some cases, heights appear too small by a factor of one hundred. For example, the correct height for `http://dbpedia.org/resource/Humberto_Contreras` would be 1.76 meters according to Wikipedia, however DBpedia gives a value of 0.0176 meters. It would seem that DBpedia expects the value to be given in centimeters and thus converts it to meters by dividing by 100.
However, in many other cases, this error is already present in Wikipedia. For example, the height of Katrina Porter (`http://en.wikipedia.org/wiki/Katrina_Porter`) is given as 1.55cm, which DBpedia then converts to 0.0155 meters accordingly.

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 0.004310m = 4.31mm at DBpedia.

5.2.3 Meter Cut Off

A number of people with actual heights of around 1.5-1.9 meters have their heights represented in DBpedia as 1.0 meter exactly. 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 `http://dbpedia.org/resource/Guy_Poitevin`.

5.3 DBpedia-owl:populationTotal

We next examined the dataset for *populationTotal*. This dataset contains 237,700 triples and we identified a total of 50 incorrect values.
The distribution of the error sources can be seen here:

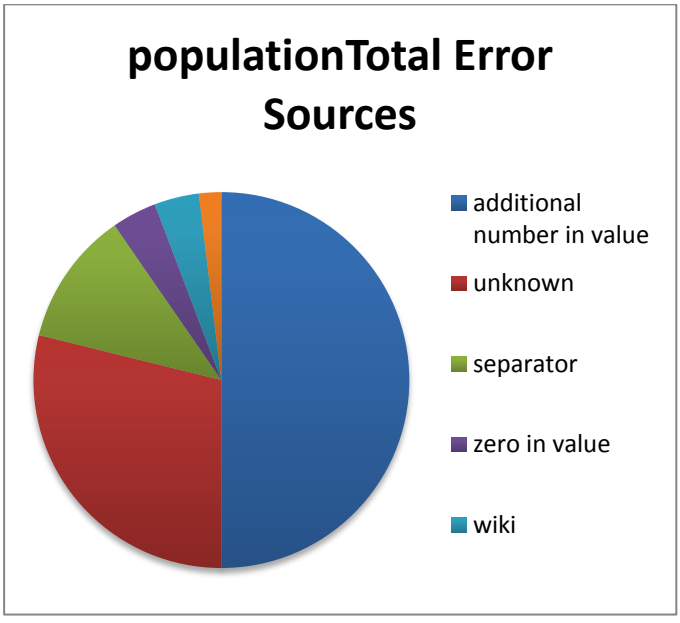

Figure 33: populationTotal error sources

5.3.1 Zero in Number

`http://dbpedia.org/resource/Durg` gives the population of the city of Durg as `2810436 (xsd:integer)`, when it really is 281,436.

A similar error can be seen with regards to the city of Nantong (`http://dbpedia.org/resource/Nantong`). DBpedia gives the population as `72828350(xsd:integer)`, when, according to Wikipedia, it is actually 7,282,835.

5.3.2 Double Information

Figure 34: Semaphore - Date in Value

The Wikipedia article for Johnstown, Colorado (http://en.wikipedia.org/wiki/Johnstown,_Colorado) gives its population as 9,887 in the infobox. However, in the introduction, a population of 3,827 in 2000 is also mentioned. In DBpedia (`http://dbpedia.org/page/Johnstown,_Colorado`), we find a combination of those two values as 38,279,887.

5.3.3 Additional Number in Value

Of the 38 incorrect resources we found, 22 had in common being small villages or towns in South Australia and their population always ending in "2006". For example `http://dbpedia.org/resource/Semaphore,_South_Australia` has a stated population of 28,322,006.
Inspection of the infobox in the original Wikipedia articles shows that the population along with the year of the census, 2006.
It appears that the "`2,832 <small><i>2006 Census</i></small>`" in the page's HTML is read as a single number, thus causing the incorrect value of 28,322,006.

5.4 Dbpedia-owl:elevation

Elevation contains 207,105 triples and we identified a total of 26 incorrect values.
The distribution can be seen here:

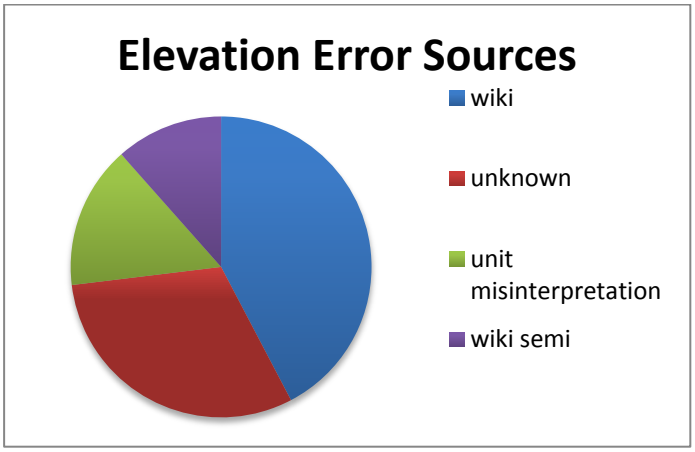

Figure 35: Elevation Error Sources

5.4.1 Unit misinterpretation

Wikipedia gives the elevation of Shadow Mountain Lake as `8367'` (8367 ft.), which DBpedia misinterprets as 8367 meters, thus causing the parsed value at `http://dbpedia.org/resource/Shadow_Mountain_Lake` to be about three times (1 meter = 3.28084 feet) larger than it actually is.

`http://dbpedia.org/resource/Zapatoca` features this error in an odd way. Wikipedia gives the elevation as `Elevation 1,720 m (4,000 ft)` and DBpedia renders this as both `1219.200000 (xsd:double)` and `13123.000000 (xsd:double)`. The first value corresponds to converting 4,000ft to meter and the latter to converting 4,000 meter to feet.

5.4.2 Wiki Semi

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

5.5 General

The following are errors that we encountered with regards to predicates other than the three we inspected in detail. As we cannot estimate the total number of errors for all datasets with reasonable accuracy, we will not be giving figures for the distribution of those errors.

5.5.1 Lists and Ranges

One of the most common errors we observed with regards to dbpprop properties concerns multiple values being concatenated into a single one.
Area codes suffer from this problem in particular because they are frequently given as lists or ranges, for example, Wikipedia gives the area code of Central California as "Area code 805, 559, 831", which DBpedia turns into `805559831 (xsd:integer)` at `http://dbpedia.org/page/Central_California` for the predicate `dbpprop:areaCode`.

5.5.2 Misinterpretation Due to Inconsistencies

Runtime

There is a property for the runtime of various media products. DBpedia extracts this from the "Running time" attribute in the corresponding infoboxes and converts the value to a common unit of time. Wikipedia does not use this property completely consistently though; for some TV shows, the runtime given is not the runtime of a single episode but the period of time over which the show aired. For example "Wielie Walie" (`http://dbpedia.org/resource/Wielie_Wielie_Walie`), a South African children's program was on for 18 years, which DBpedia interprets as a runtime of $18 \cdot 365 \cdot 24 \cdot 60 \cdot 60 \approx 5.68 \cdot 10^8$ seconds.

Another problem with regards to the runtime occurs when Wikipedia gives the running time in meters (of film), which then gets interpreted as minutes. For example, the runtime of the 1919 movie "The Unpardonable Sin" is given as "9 reels (2,700 meters)" on Wikipedia, which then gets converted to 162,000 second = 2,700 minutes at`http://dbpedia.org/resource/The_Unpardonable_Sin_(1919_film)` for `http://dbpedia.org/ontology/runtime`.

The `http:// dbpedia.org/ontology/activeYearsEndYear` and `http:// dbpedia.org/ontology/activeYearsStartYear` properties suffer from similar problems when a timespan is given and misinterpreted as a date. For example, `http://en.wikipedia.org/wiki/Lions_Gate_Chorus` has a "Years active" attribute with a value of 53, which `http://dbpedia.org/resource/Lions_Gate_Chorus` interprets as the year 53.
In some cases, however, this is caused by misinterpreting a decade given at Wikipedia, such as "Mid-90s – present" for `http://en.wikipedia.org/wiki/Depswa` as an absolute year.

Elevation Ballistics vs. Geographical

A similar case of misinterpretation due to double meaning can be observed with regards to `dbp-prop:elevation`, where for resources of type `dbpedia-ontology:weapon` ballistic elevation (the angle between the barrel and the ground) has been misinterpreted as geographic elevation.

5.5.3 Date in value

Some songs have a date in the runtime field for a specific version. For example the Wikipedia article for "White Christmas" features this in its infobox:

Length	3:02 (1942recording) 3:04 (1947 recording)

This is then misinterpreted as a runtime of 1942 (seconds) at `http://dbpedia.org/resource/White_Christmas_(song)` for `http://dbpedia.org/ontology/Work/runtime`. Other examples are `http://dbpedia.org/resource/The_First_Time_Ever_I_Saw_Your_Face`,`http://dbpedia.org/resource/The_Singer_Sang_his_Song` and `http://dbpedia.org/ontology/Work/runtime`.

5.5.4 Separators

DBpedia will sometimes assume dots to represent decimal separators and commas to represent thousands separators, so if this assumption is violated, errors occur.

5.5.5 Comma As Decimal Separator

Commas are sometimes interpreted as thousands separators, so when a comma is used to indicate the decimal point, it gets lost, e.g. for the property `http:// dbpedia.org/property/ eccentricity` and the subjects `http:// dbpedia.org/resource/1134_Kepler`, `http:// dbpedia.org/resource/2644_Victor_Jara` and `http:// dbpedia.org/ resource/266983_Josepbosch`.

5.5.6 Dot as Thousands Separator

If a dot is used as a thousands separator it is interpreted as a decimal separator and especially when the value is expected to be an integer, such as for the population of `http://dbpedia.org/ resource/Garg%C5%BEdai` , a city with a population of "16.814", which becomes a population of 17 after misinterpretation and rounding.

5.5.7 Comma as Thousands Separator

This behavior is not consistent though and we have seen the interpretation going wrong the other way around. `http://dbpedia.org/resource/Fruitland,_Maryland` states a population of 5, when in reality it is 4,866.

5.5.8 Time Misinterpretation

Wikipedia lists the runtime of some albums as mm:ss:msms, for example `http:// en.wikipedia.org/wiki/Les_Dudek_(album)` . DBpedia reads this as hh:mm:ss and converts it to `"2589.1666666666665"^^http://dbpedia.org/datatype/minute"` at `.http:// dbpedia.org/page/Les_Dudek_(album)` .

5.6 Dbpprop vs. OWL

DBpedia features two kinds of ontologies: *dbpprop* (`http://dbpedia.org/property`) and *dbpedia-owl* (`http://dbpedia.org/ontology`) . *Dbpprop* aims at high coverage and thus includes a wide range of different properties with various units at the expense of proper range and domain specifications. (The DBpedia Data Set, 2013)*Dbpedia-owl* on the other hand aims at consistency and higher data quality.

We have seen this during our evaluation where we found thousands of incorrect values for some *dbpprop* properties while errors in the *dbpedia-owl* properties were much more spurious and much harder to come by.

5.6.1 Data types

We first took a look at the available properties for each ontology using the 50 random resources that we already used for evaluation earlier on. This yielded 121 different properties for *dbpedia-owl* and 315 for *dbpprop*.

DBpedia-OWL

Next, we inspected the data types of the corresponding objects. 63 of the 121 *dbpedia-owl* properties were of type `ObjectProperty` and thus the objects are URIs that cannot have a data type. 19 of the remaining 58 (~33%) *dbpedia-owl* properties did not feature a single object with a defined data type. However, 18 of those had a range of string so a data type would not be necessary.

The only property that appeared inconsistent regarding the declared range and the actual data types of the objects was `http://dbpedia.org/ontology/individualisedPnd`, which represents "PND (Personennamendatei) data about a person. PND is published by the German National Li-

brary. For each person there is a record with her/his name, birth and occupation connected with a unique identifier, the PND number." This property has a declared range of `xsd:nonNegativeInteger`, however all objects are German language strings. The reason appears to be that in some cases, the last digit is replaced by an "X", which would not be possible with true integers.

DBprop

DBpprop properties are all only of type `rdf:property` so there is no indication of whether they are supposed to have literals or other resources as objects. They do not feature declared ranges either, so we have to go completely by what objects we find. 21 of the 315 (6.7%) properties did not have any objects with data types. We found 76 different data types for *dbpprop*, including exotic currencies such as the Ghanaian Cedi (`http://dbpedia.org/datatype/ghanaianCedi`) or the Nicaraguan córdoba (`http://dbpedia.org/datatype/nicaraguanC%C3%B3rdoba`) and metric units with rarely used prefixes such as the giga meter (`http://dbpedia.org/datatype/gigametre`) or the mega liter (`http://dbpedia.org/datatype/megalitre`).

In contrast, the data types for *dbpedia-owl* do appear quite consistent. There are only 15 of them, 7 of which are sourced from w3.org.

Further corroborating the consistency of *dbpedia-owl* as compared to *dbpprop* is that of the 121 properties 115 (95%) featured exclusively objects of a single type and only 6 (5%) properties featured objects of two different types.

For *dbpprop* we found the following distribution regarding the frequency of different data types:

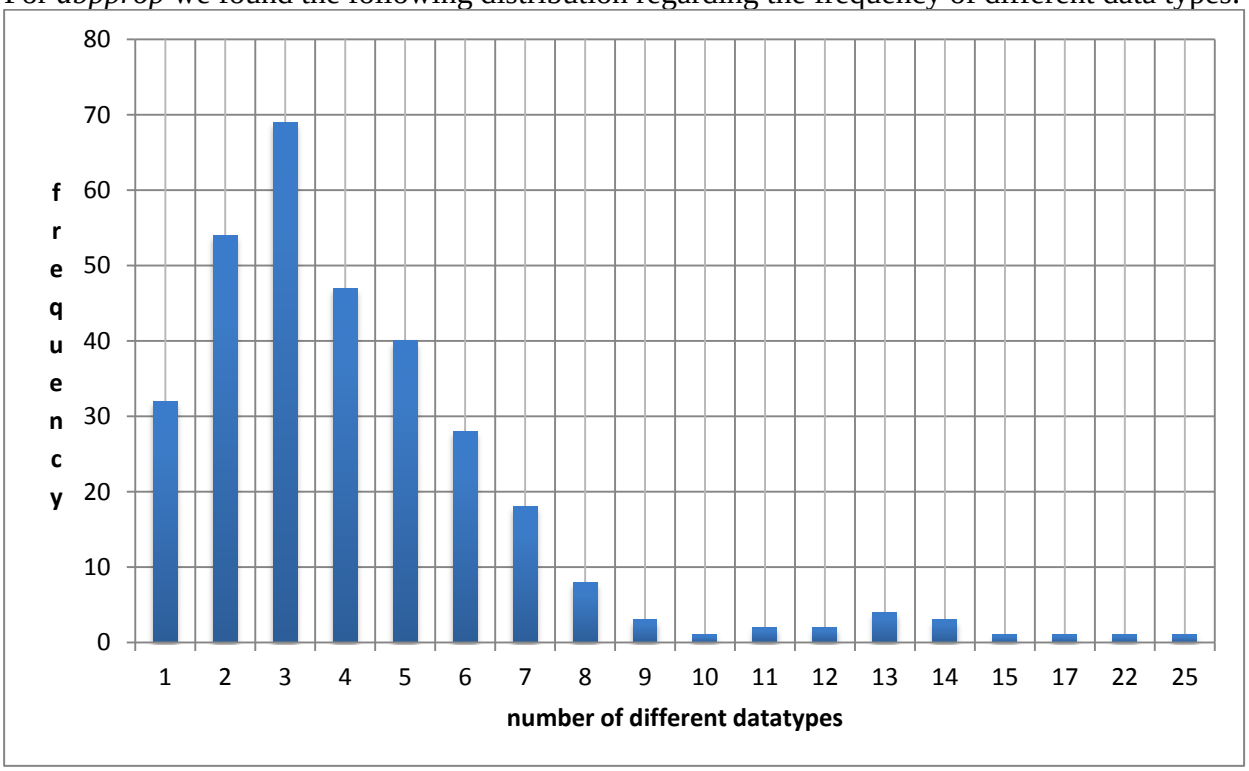

Figure 36: number of data types distribution

There is one property, `http://dbpedia.org/property/altNames`, which features objects of 25 different types. This would obviously make automatic processing extremely complicated. How-

ever, even dealing with the most common case of three different data types would mean a large amount of additional work.

Data types of `http://dbpedia.org/property/altNames` objects:

data type	count
http://www.w3.org/2001/XMLSchema#integer	647
http://www.w3.org/2001/XMLSchema#int	10820
	673
http://dbpedia.org/datatype/terabyte	10
http://dbpedia.org/datatype/joule	1
http://dbpedia.org/datatype/newton	2
http://dbpedia.org/datatype/kelvin	2
http://dbpedia.org/datatype/degreeRankine	2
http://dbpedia.org/datatype/volt	2
http://dbpedia.org/datatype/astronomicalUnit	4
http://dbpedia.org/datatype/usDollar	6
http://dbpedia.org/datatype/gigalitre	4
http://dbpedia.org/datatype/gigabyte	9
http://dbpedia.org/datatype/giganewton	7
http://dbpedia.org/datatype/pferdestaerke	5
http://dbpedia.org/datatype/meganewton	2
http://dbpedia.org/datatype/megalitre	2
http://dbpedia.org/datatype/megabyte	7
http://dbpedia.org/datatype/megawatt	2
http://dbpedia.org/datatype/gigawatt	4
http://dbpedia.org/datatype/second	2
http://dbpedia.org/datatype/megavolt	2
http://dbpedia.org/datatype/litre	1
http://dbpedia.org/datatype/degreeFahrenheit	2
http://dbpedia.org/datatype/ampere	1

Table 7: altNames data types

This is another example of a predicate that is obviously meant to have integer objects, as 22 of the data types only appear in an extremely small fraction of the cases.

5.6.2 Property Comparison

The three specific predicates, *height*, *populationTotal* and *elevation*, which we already used for evaluation earlier, are not only available in *dbpedia-owl* but in *dbpprop* as well, so we used those pairs to performs some comparisons.

Elevation

The *dbpprop* version of the *elevation* dataset contains only 13,162 data points, compared to 206,997 for the *dbpedia-owl* version. Yet we found 178 resources with elevations greater than that of Mount Everest (8,848 meters). We further found 501 resources of type `dbpedia-ontology:weapon` where obviously the ballistic elevation (the angle between the barrel and the ground) had been mis-

interpreted as geographic elevation. That makes for a total of 679 easily identifiable errors (5%) as opposed to only 28 (0.01%) in the *dbpedia-owl* version.

Height

For *height*, there are 14,382 triples in the *dbpprop* version and 52,252 in the *dbpedia-owl* version. But beyond that, comparison is much more difficult. While the *default* unit for `dbpedia-owl:height` is defined as meter, for `dbpprop:height` objects are of many different data types on the one hand and of type xsd:int or completely untyped on the other, which makes it impossible to accurately infer a single *default* unit.

Valid units of length used in the `dbpprop:height` dataset are `http://dbpedia.org/datatype/millimetre` and `http://dbpedia.org/datatype/inch` (mostly cars), `http://dbpedia.org/datatype/centimetre` (persons) `http:// dbpedia.org/datatype/foot`, `http:// dbpedia.org/datatype/kilometre` and `http:// dbpedia.org/datatype/astronomicalUnit`. However, objects of those data types only make up 3,267 of all objects (22.7%), so further analysis does not seem viable.

populationTotal

For *populationTotal*, both versions are similar in size (237,700 for dbpedia-owl vs. 164,407 for dbpprop). However, we were not able to identify any significant errors in the *dbpprop* version beyond those that occur in the *dbpedia-owl* version too.

50 Random Resources

We were not able to examine all 50 datasets in detail so we cannot make an accurate estimate of the error ratios for all 50 datasets. However, of the 9,039 outliers we detected, only 1009 (11.2%) belonged to the *dbpedia-owl* namespace and the remaining 8,030 (88.4%) belonged to the *dbpprop* namespace.

6. Conclusion

6.1 Results

6.1.1 Methods

The simple IQR method delivers some of the best results in our tests. Combined with the *byType* mode, we achieved about 87% precision on small high quality samples as well as on large random samples. Multipliers of 30-50 and quantiles of about 0.006 to 0.011 yield the best results in our manually evaluated tests. However, those parameters are still dependent on the dataset under inspection, as can be seen with regards to the tests on random samples.

Basic KDE shows similar results with relative probability thresholds of 0.3 to 0.4 but suffers from high runtimes. A high threshold of 0.93 to 0.94 allowed KDE to discover outliers that no other method could identify with similar precision. However, this result is very specific to the *height* dataset and even then precision falls below 50%.
The other methods, *dispersion* and KDE-FFT, mostly fail to deliver results with more true than false positives.

In order to be able to use a single method with a single set of parameters for a large amount of different datasets it has to be extremely robust and work with a large safety margin. Without this safety margin, one dataset with a large number of natural outliers can often cause more false positives than what would be feasible to inspect manually. This again means, however, that recall will generally low because once the algorithm begins to pick up false positives, it is generally not possible to compensate by increasing the sensitivity as that would only cause more false positives, in addition to the ones that already exist.

6.1.2 Modes

Regarding semantics, the biggest improvement upon the baseline can be achieved by splitting the datasets by RDF types.
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 *byType* or *cluster* and the combinations of KDE, *byType* or *cluster* and *iterative* produce the best results.

6.1.3 DBpedia Analysis

We identified a number of common sources of errors in DBpedia. Large amounts of the corrupted data points 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.

Direct comparisons between *dbpedia-owl* and *dbpprop* do not seem particularly meaningful because they aim for completely different goals. While *dbpedia-owl* is based on consistency with declared ranges and domains, *dbpprop* aims for as much coverage as possible. Thus, when comparing the two, we are faced with comparing a mostly correct and consistent dataset with one, where we can

often not even determine what the data is meant to represent. However, to put them into some relation: of the outliers we inspected in our random sample, 88% came from the *dbpprop* namespace and only the remaining 12% from *dbpedia-owl*.

6.2 Outlook

6.2.1 Integrated Implementation

Our implementation is mainly optimized for the test tracks we built for evaluation.
We did not use a complete local copy of DBpedia and to compensate we employed disk and memory caches to speed up repeated operations on the same data.
This does not reflect the regular use case where all the data is locally available but only processed once.
If outlier detection were deployed alongside a full copy of DBpedia some variables would change.
Performance-wise, we do not see much room for improvement on most basic analysis methods (except KDE) but the advanced methods that actively utilize Semantic Web features could certainly profit from having all data available locally.
If outlier detection has to be done on the fly, probably only IQR (with *default* or *byType* mode) would be feasible. If, on the other hand, the outlier detection could run as a background task that uses free processing power, even the more time-consuming methods and modes such as KDE and cluster could become feasible.

6.2.2 Frontend Integration

Outlier detection and particularly subsequent parsing could be integrated on the fly into LOD frontends to allow users to judge the reliability of triples and possibly show alternative values.

We created a simple implementation of the 4.15 majority data type approach with subsequent parsing in the MoB4LOD framework (MoB4LOD, 2012) that would mostly be beneficial for developers.
Figure 37: Majority Data type Parsing MoB4LOD shows an example screenshot of this implementation, displaying a snippet of the results for the property `dbpprop:aprRecordLowC`. In the lower half, we see (a small part of) the triples featuring the majority data type `xsd:int`. At the top, we see triples that were deemed incorrect because they were given as strings and subsequently parsed to the majority data type of integer.

http://dbpedia.org/resource/Lethbridge	http://dbpedia.org/property/aprRecordLowC	~~-25.6@en~~ -25^^http://www.w3.org/2001/XMLSchema#int
http://dbpedia.org/resource/Ciudad_Obregón	http://dbpedia.org/property/aprRecordLowC	~~-2@en~~ -2^^http://www.w3.org/2001/XMLSchema#int
http://dbpedia.org/resource/Valdivia	http://dbpedia.org/property/aprRecordLowC	~~-2@en~~ -2^^http://www.w3.org/2001/XMLSchema#int
http://dbpedia.org/resource/Miramichi,_New_Brunswick	http://dbpedia.org/property/aprRecordLowC	~~-17.2@en~~ -17^^http://www.w3.org/2001/XMLSchema#int

int data

Number of triples: 1339

Subject	Predicate	Object
http://dbpedia.org/resource/Geography_of_Ireland	http://dbpedia.org/property/aprRecordLowC	-4^^http://www.w3.org/2001/XMLSchema#int
http://dbpedia.org/resource/Lyon	http://dbpedia.org/property/aprRecordLowC	-4^^http://www.w3.org/2001/XMLSchema#int
http://dbpedia.org/resource/Darmstadt	http://dbpedia.org/property/aprRecordLowC	-4^^http://www.w3.org/2001/XMLSchema#int
http://dbpedia.org/resource/Invergowrie	http://dbpedia.org/property/aprRecordLowC	-4^^http://www.w3.org/2001/XMLSchema#int
http://dbpedia.org/resource/Bangor,_County_Down	http://dbpedia.org/property/aprRecordLowC	-4^^http://www.w3.org/2001/XMLSchema#int
http://dbpedia.org/resource/Cerro_El_Pital	http://dbpedia.org/property/aprRecordLowC	-4^^http://www.w3.org/2001/XMLSchema#int
http://dbpedia.org/resource/Zagreb	http://dbpedia.org/property/aprRecordLowC	-4^^http://www.w3.org/2001/XMLSchema#int
http://dbpedia.org/resource/Cusco	http://dbpedia.org/property/aprRecordLowC	-4^^http://www.w3.org/2001/XMLSchema#int
http://dbpedia.org/resource/Tlell,_British_Columbia	http://dbpedia.org/property/aprRecordLowC	-4^^http://www.w3.org/2001/XMLSchema#int
http://dbpedia.org/resource/Young,_New_South_Wales	http://dbpedia.org/property/aprRecordLowC	-4^^http://www.w3.org/2001/XMLSchema#int

Figure 37: Majority Data type Parsing MoB4LOD

6.3 Semantics

By using the type information of the resources, we have only scratched the surface of the semantics in Linked Data. As we (partially) explained in the introduction, the Semantic Web provides for an incredibly rich theoretical foundation that could potentially be utilized for outlier detection.

The two main problems are:

1. Actual implementations are still lacking far behind what is theoretically possible.
2. The lower the general quality of the data - and thus the greater the need for outlier detection - the fewer semantics tend to be available.

However, once a basis of consistent data, which is properly augmented with semantics, has been created, outlier detection for further data could be vastly improved.

6.3.1 Semantics Error Detection

Another path could be to extend the outlier detection to the semantics themselves. So, instead of taking the semantics, such as types and ranges, for granted and using them to look for errors in the numerical data, one could work in the opposite direction and use the numerical data to detect errors in the semantics. For example, if an entity of type "village" has a population of over one million, do not necessarily assume the population to be wrong but instead the type.

6.3.2 Clustering

We evaluated one clustering approach on RDF types, which showed promising results but suffered from extremely high runtimes. However, there is a myriad of different clustering algorithms and

there is much more information available for each subject beyond its type that could be used in the clustering process. Other clustering algorithms could be evaluated for better runtime performance on RDF types and further attributes could be combined with various clustering algorithms to achieve more useful clustering results.

6.4 *Linked* Data

By studying DBpedia, we have actually just examined a single point of the Linked Data cloud in isolation. A natural expansion to outlier detection in Linked Data would be to utilize the linkage inherent in the data. By comparing suspicious values corresponding values from other sources, it could not only be possible to detect more outliers but to correct them automatically too.
Another approach could be to use a Linked Data source that features low quantity but high quality data to build training data for (semi) supervised outlier detection schemes on lower quality but higher quantity sources such as DBpedia.

6.5 Univariate Outlier Detection Methods

The numerical outlier detection methods we inspected and used are fairly simple but development has recently only been made in the area of multivariate outlier detection schemes. IQR and the *dispersion* methods would obviously be unable to cope with bimodal distributions. Basic KDE does not seem feasible for large datasets while the FFT supported version is, at least in its naïve implementation, not precise enough for our purposes. However, seeing how promising the results of KDE are, a middle ground between performance and precision could be the silver bullet.

References

MoB4LOD. (2012). Retrieved from ke.tu-darmstadt.de: http://www.ke.tu-darmstadt.de/resources/mob4lod

The DBpedia Data Set. (2013, 2 22). Retrieved 5 25, 2013, from dbpedia.org: http://wiki.dbpedia.org/Datasets?v=bli

The R Project for Statistical Computing. (2013). Retrieved 5 26, 2013, from r-project.org.

Berners-Lee, T. (2006, 7 27). *Linked Data*. Retrieved 5 27, 2013, from w3.org: http://www.w3.org/DesignIssues/LinkedData.html

Bizer, C. (2012, 8 6). *DBpedia 3.8 released, including enlarged Ontology and additional localized Versions*. Retrieved 5 24, 2013, from DBpedia.org: http://blog.dbpedia.org/2012/08/06/dbpedia-38-released-including-enlarged-ontology-and-additional-localized-versions/

Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., & Hellmann, S. (2009). DBpedia - A Crystallization Point for the. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*(7), 154-165.

Brickley, D., Guha, R., & McBride, B. (2004, 2 10). *RDF Vocabulary Description Language 1.0: RDF Schema*. Retrieved from w3.org: http://www.w3.org/TR/rdf-schema/

Chandola, V., Banerjee, A., & Kumar, V. (2009, 7). Anomaly detection: A survey. *ACM Computing Surveys (CSUR), 3*(41).

Chauvenet, W. (1863). *A Manual of Spherical and Practical Astronomy V. II*. Philadelphia: J. B. Lippincott & co.

Cyganiak, R., & Jentzsch, A. (2011, 9 19). *The Linking Open Data cloud diagram*. Retrieved 5 24, 2013, from lod-cloud.net: http://lod-cloud.net/

Dean, R. B., & Dixon, W. J. (1951). Simplified Statistics for Small Numbers of Observations. *Analytical Chemistry, 23*(4), 636-638.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological), 1*(31), 1-38.

Estivill-Castro, V. (2002, 6). Why so many clustering algorithms — A Position Paper . *ACM SIGKDD Explorations Newsletter, 1*(4), 65-75.

Fleischhacker, D. (2013). *Detecting Data Errors in Linked Data*.

Gauss, F. (1816). Bestimmung der Genauigkeit der Beobachtungen. *Zeitschrift für Astronomie und verwandte Wissenschaften*, 195.

Grubbs, F. (1950). Sample Criteria for Testing Outlying Observations. *Annals of Math. Statistics, 21*, 27-58.

Grubbs, F. (1969, 2). Procedures for Detecting Outlying Observations in Samples. *Technometrics, 11*(1), 1-21.

Härdle, W., Müller, M., Sperlich, S., & Werwatz, A. (2004, 6 9). *Nonparametric and Semiparametric Models*. Retrieved 5 25, 2013, from wiwi.hu-berlin.de: http://sfb649.wiwi.hu-berlin.de/fedc_homepage/xplore/ebooks/html/spm/

Hawkins, D. (1980). *Identification of Outliers*. London: Chapman and Hall.

Klyne, G., Carroll, J., & McBride, B. (2004, 2 10). *Resource Description Framework (RDF): Concepts and Abstract Data Model*. Retrieved from w3.org: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

L'Ecuyer, P. (2012, 2 16). *SSJ: Stochastic Simulation in Java*. Retrieved 5 25, 2013, from iro.umontreal.ca: http://www.iro.umontreal.ca/~simardr/ssj/indexe.html

Maronna, R. A., & Zamar, R. H. (2002). Robust Estimates of Location and Dispersion for High-Dimensional Datasets. *Technometrics, 4*(44), 307-317.

Marsaglia, G., & Bray, T. A. (1964, 7). A Convenient Method for Generating Normal Variables. (S. f. Mathematics, Ed.) *SIAM Review, 3*(6), 260-264.

Parzen, E. (1962). On Estimation of a Probability Density Function and Mode. *The Annals of Mathematical Statistics, 3*(33), 1065-1076.

Paulheim, H., & Fürnkranz, J. (2012). Unsupervised Generation of Data Mining Features. *WIMS'12*. Craiova: ACM.

Peirce, B. (1852, 7 24). Criterion for the Rejection of Doubtful Observations. *Astronomical Journal, 2*(45).

Rosenblatt, M. (1956). Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics, 3*(27), 832-837.

Rousseuw, P. J., & Croux, C. (1993, 12). Alternatives to the Median Absolute Deviation. *Journal of the American Statistical Association, 424*(88), 1273-1283.

Shannon, V. (2006, 5 23). *A 'more revolutionary' Web - The New York Times*. Retrieved 5 24, 2013, from The New York Times: http://www.nytimes.com/2006/05/23/technology/23iht-web.html

The University of Waikato. (2013). *Weka 3: Data Mining Software in Java*. Retrieved 5 25, 2013, from cs.waikato.ac.nz: http://www.cs.waikato.ac.nz/ml/weka/index.html

Village.rdf. (n.d.). Retrieved 5 24, 2013, from umbel.org: http://umbel.org/umbel/rc/Village

W3C. (2013). *SEMANTIC WEB*. Retrieved 5 27, 2013, from w3.org: http://www.w3.org/standards/semanticweb/

W3C OWL Working Group. (2012, 12 11). *OWL 2 Web Ontology Language* . Retrieved from w3.org: http://www.w3.org/TR/owl2-overview/

Zaveri, A., Kontokostas, D., Sherif, M. A., Morsey, M., Bühmann, L., Auer, S., & Lehmann, J. (2012). *Crowd-sourcing the Evaluation of Linked Data*. Leipzig.

List of Figures

List of Tables

Erklärung

Hiermit versichere ich gemäß der Allgemeinen Prüfungsbestimmungen der Technischen Universität Darmstadt (APB) §23 (7), die vorliegende Bachelorarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 3. Juni 2013 **Dominik Wienand**