On Reliability Scores for Knowledge Graphs

Thomas Grubb
University of California, San Diego*
San Diego, California, USA
tgrubb@ucsd.edu

Bill Andersen
Instacart
San Francisco, California, USA
bill.andersen@instacart.com

Omar Alonso
Instacart
San Francisco, California, USA
oralonso@gmail.com

ABSTRACT

The Instacart KG is a central data store which contains facts regarding grocery products, ranging from taxonomic classifications to product nutritional information. With a view towards providing reliable and complete information for downstream applications, we propose an automated system for providing these facts with a score based on their reliability. This system passes data through a series of contextualized unit tests; the outcome of these tests are aggregated in order to provide a fact with a discrete score: reliable, questionable, or unreliable. These unit tests are written with explainability, scalability, and correctability in mind.

CCS CONCEPTS

• **Information systems → Graph-based database models**.

KEYWORDS

knowledge graphs, data cleaning, reliability

ACM Reference Format:
Thomas Grubb, Bill Andersen, and Omar Alonso. 2022. On Reliability Scores for Knowledge Graphs. In *Companion Proceedings of the Web Conference 2022 (WWW '22 Companion), April 25–29, 2022, Virtual Event, Lyon, France.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3487553.3524212

1 INTRODUCTION

A *knowledge graph* (KG) is a collection of entities together with statements connecting these entities. Knowledge graphs are increasingly used by large companies as a means for organizing and understanding their data. At Instacart we designed and implemented a product knowledge graph to understand inventory items in our grocery catalogs. The entities we use are thus often food products, classes of food products such as *Pasta* (which are governed by an internal taxonomy), brands of products such as *Coca-Cola*, or product related attributes, such as flavors or nutritional information. Statements regarding these entities express relationships between them, and will be written in SPO *(subject, predicate, object)* format; thus *(P, brand, Coca-Cola)* means that *P* is a Coca-Cola branded entity, and *(P, type, Vegan)* means that *P* is a Vegan entity.

Modern product knowledge graphs are derived from numerous sources and contain a wealth of information. The Instacart KG contains information regarding products, recipes, and various product attributes, together with millions of contextual facts regarding these entities. As of 2019, Microsoft's knowledge graph contained approximately 55B facts regarding 2B entities, and Google's knowledge graph contained 70B facts regarding 1B entities [5]. When orchestrated correctly, these graphs can provide value to search, online advertisement, question-answering, and recommendation efforts [4].

Due to their large scale it is infeasible to curate such graphs by hand. Because of this, automated quality control mechanisms are important to ensure KGs contain valid information. Often KGs are created through a series of automated ETL processes which analyze both structured and unstructured data from a variety of sources to generate facts for the graph. This automation, combined with questionable source data, can cause KGs to acquire noise in the form of incorrect statements during their build processes. This noise can present itself in a variety of ways: incorrect product attributes can lead to negative storefront interactions, and noisy training sets can lead to less precise machine learning models. This has led to much work regarding quality assessment, error detection, and error correction in knowledge graphs [4, 7].

In this article we will discuss a range of techniques that we employ to give reliability scores for data in our KG[1]. The system relies on a series of practical, white-box unit tests which examine particular slices of the knowledge graph. When aggregated, these tests assign facts one of three scores: reliable, questionable, or unreliable.

Scoring our data in this fashion yields two major benefits. First, the discrete thresholds yield guardrails for data publication. It is trivially easy to restrict a KG query to only select data which is at or above a certain reliability score. Second, the white box nature of our unit tests allows us to easily examine noise in our data. Each unit test produces an error log during runtime; these error logs can be aggregated and examined to look for hot spots of noise in source data. These errors can then be accounted for at the source to result in a cleaner KG in subsequent builds.

2 INFRASTRUCTURE

The Instacart KG is managed through Amazon's AWS Neptune platform. Information in the KG is stored using RDF triples, and the KG is queried using the SPARQL query language. Initialization of the graph occurs via bulk loading of triples, and analysis and editing after this initialization can be done via SPARQL select and update statements. We also have a workflow in place that automates the generation of the triples and the bulk upload making it easier to generate the graph from scratch within a few hours.

*Research conducted during PhD Internship at Instacart.

[1] As the Instacart KG contains proprietary information, we will abstract away specific implementation details and results where necessary.

WWW '22 Companion, April 25–29, 2022, Virtual Event, Lyon, France.

ACM ISBN 978-1-4503-9130-6/22/04…$15.00
https://doi.org/10.1145/3487553.3524212

The KG is continually rebuilt from an ever growing set of seed data which comes from a variety of (ever growing) sources. This build process standardizes, centralizes, and enriches this seed data to create a staging KG. The standardization and contextualization that occurs during the build process allows for more uniform treatment of data; for example, *strings* representing product sizes are transferred to a numeric literal together with a recognized unit of measure from then Quantities, Units, Dimensions, and Types (QUDT) ontology. As a result of this, reliability assessment occurs *after* the staging KG has been built.

Once the staging KG has been built the scoring can occur. Our reliability system currently tests one slice of the KG, such as our knowledge of product nutritional information, at a time. To test these slices we first query the relevant data using SPARQL to obtain a local, tabular form of the information (in practice, each class of test comes with its own SPARQL query which prepares the relevant data for that test). The queried data is then run against a library of unit tests. These unit tests are coded as Python Classes. Each test comes with both an evaluation step, which tests the data, and a logging step, which logs the result.

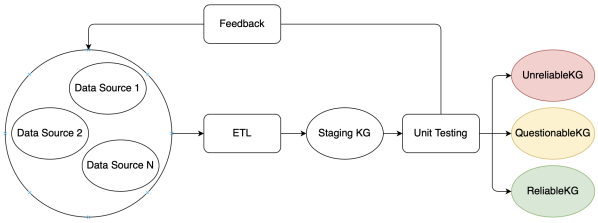

Figure 1: A high level overview of the KG scoring process

After these tests have been run each piece of data is tagged with a list of tests that failed. Aggregating this information allows us to tag that piece of data with a score of reliable, questionable, or unreliable. At this point the process forks into a feedback mechanism and an update mechanism. The feedback mechanism logs the results of the unit testing. These logs can be passed upstream to data providers to clean data. The update mechanism uses SPARQL Insert statements to update data into one of three named graphs in Amazon Neptune: the ReliableKG, the QuestionableKG, and the UnreliableKG.

As a concrete example of how this fits into the data curation process: questionable product-taxonomy associations may be obtained by running tests described in Section 3.3 and 3.4. If data is being published for a targeted downstream use case (e.g., a holiday promotion) we may restrict this report to the products related to that use case; this shortened report can then be used for minor spot checks prior to public facing uses. Alternatively, the full report may be sent out to data annotators to validate or correct source catalog data in bulk. In doing so, we focus costly data annotation efforts towards the areas of the catalog which need it most.

While the use of discrete scores and named graphs does have restrictions, this infrastructure was developed with practicality and ease of implementation in mind. Querying specific named graphs in SPARQL is achieved by simply tagging the query with the graph (or graphs) of interest. Additionally, untagged queries in Neptune run against the *default graph* (the union of all named graphs), which means that all existing queries that ran against the KG pre-scoring will run *without modification* against the post scored graph. An alternative method for implementing edge scoring, such as RDF reification, would allow greater flexibility, but would introduce additional technical overhead. To query for edge scores that were stored using reification would require more verbose SPARQL queries, since storing one piece of edge metadata using reification requires the use of four RDF triples.

3 RELIABILITY TESTS

In this section we will discuss several types of test that are used to score data in the KG. We reemphasize that these tests are designed with interpretability and practicality in mind. The use of techniques such as graph embeddings can lead to very powerful KG cleaning techniques; however, an overreliance on machine learning for computing KG reliability scores can result in black box techniques that are hard to interpret and learn from. These techniques additionally come with large implementation and upkeep costs.

Because of this, we have decided to start our reliability efforts with explainable, easy to implement unit tests. This has allowed us to quickly establish a baseline infrastructure for KG reliability which provides actionable feedback on data quality in the short term and which can further be improved on in the long term. We have grouped the unit tests according to their underlying structure into five groups below, which we will discuss in turn.

3.1 Numerical

The purpose of numerical testing is to apply quick sanity checks to numeric literals in our KG. Simple versions of these tests include applying thresholds to data, such as requiring that numeric literals be non-negative. The more widely applicable form of this is outlier detection.

Outlier detection schemes can rely on a variety of methods, such as the use of interquartile ranges or kernel density estimation. See [12] for a good summary. As noted in the cited work, although outlier detection can occasionally *incorrectly* flag true outliers in numeric data, outliers in data are frequently due to actual errors due to things such as typos.

As discussed in [1], outlier detection schemes greatly benefit from restricting the set of products examined at a single time to a natural class. For example, when we run outlier detection on our nutritional information. we only test a single taxonomy class, such as *Peaches*, at a time. Doing so provides a much higher recall rate for inaccuracies, with limited drop in precision. Restricting the class of products tested at a single time also has the effect of working as an implicit test of a product's taxonomy class. A can of tuna T with correct nutritional information and with incorrect classification $(T, type, Peach)$ will be flagged by an outlier detection scheme, as the protein in a can of tuna is much higher than that in a peach.

While multivariate outlier detection schemes are possible to use for quality testing, benefits to using univariate schemes include specificity and explainability. In addition to this, univariate outlier detection schemes can serve a dual purpose as attribute creation techniques. Products that are true outliers with respect to certain numeric literals can be verified and tagged with attributes such as "bulk", "high-protein", or "low-calorie" to aid in search retrieval.

3.2 Constraint Based

The use of *rules and constraints* is a common technique for knowledge discovery and knowledge cleaning in KGs [11]. The use of rules and constraints involves invoking domain specific ground truths (called *intensional knowledge*) to certain data in the knowledge base. This intensional knowledge creates invariants of the graph; failure of these invariants is then a red flag for inconsistency.

Frequently constraint based testing can be seen as an extension of the graphs ontology. For example, *inclusion based* constraints are based on subclass relationships. As an example,

$$\forall x, (x, type, Vegan) \implies (x, type, Vegetarian).$$

Intensional knowledge is not limited to the ontology of a knowledge graph. The use of *value base* testing can be used to give numeric restrictions on the graph. Two basic examples are

$$\forall x, v, (x, type, sugarfree) \wedge (x, sugarPerServing, v) \implies v = 0.$$

and

$$\forall x, v_1, v_2, (x, carbsPerServing, v_1) \wedge sugar(x, sugarPerServing, v_2) \implies v_1 \geq v_2.$$

As the information in our KG largely revolves around *food* products, we can obtain nice mileage out of several bespoke constraints derived from domain specific knowledge of food and nutrition.

In addition to the constraints mentioned above which *must* be satisfied for consistency, one can invoke *soft rules* which describe the *usual* behaviour of consistent data. Violations of soft rules are not always due to mistakes in data, but often should be treated with caution. An easy example of a soft rule that can be applied to product knowledge graphs involves branding. For example, most Sprite products are beverages:

$$\forall x, (x, brand, Sprite) \implies (x, likelyHasType, Beverage).$$

Modern knowledge bases often perform *automated rule mining* with the goal of discovering soft rules; see RUDIK [6] for an example.

3.3 Clustering

Although KGs think in terms of "things, not strings", it can be useful to remember that strings often carry semantic information regarding an entity. We use this idea to relate products to taxonomy classes via their titles.

Various methods for doing related tasks exist in the literature. For example, Gharibi et al. [3] employ ConceptNet [10] to assign a class of items with several "representative" entities

$$Fruit \rightarrow \{mango, strawberry, \dots\}$$

A string is then assigned to the class which minimizes the average distance from the string to the representative entities, based on word embeddings. Their approach uses a fixed number of representative entities per class. Due to the wide variety of products and product titles in our catalog, we chose a different approach based on the k-nearest neighbors algorithm.

Using a word embedding model, such as GloVe[8], we obtain a product to vector embedding by averaging the word embeddings of a product's title:

$$(P, title, \text{"Hard Apple Cider"}) \implies v_P = \frac{v_{hard} + v_{apple} + v_{cider}}{3}.$$

We then run a modified k-NN algorithm to map v_P to a set of neighbors:

$$v_P \rightarrow \{v_{Q_1}, \dots, v_{Q_k}\}.$$

We then compare the taxonomy class of P to that of $Q_1, \dots, Q_k$. If the taxonomy class of P is not shared by any Q_i we see this as a sign of unreliability; in this case we lower the confidence of the taxonomy class of P and use the classes of $Q_1, \dots, Q_k$ as a signal towards a possible correction.

To allow this to scale, we first downsample the search space using a preprocessing step. This preprocessing step creates a hash which maps a set of words to the list of products which contain all of those words in their title:

$$\{Apple, Cider\} \rightarrow [P_{141}, P_{985}, \dots].$$

When we then look for neighbors of v_P, we can then first map P to subsets of words in the product's title. Downsampling occurs by restricting the search space to products which share the most words in common with P.

3.4 Anomaly Detection

Anomaly detection relies on relating an entity to similar entities in the KG and then comparing and contrasting their attributes. This can be a useful method for detecting categorical inaccuracies; we apply such tests to brand and taxonomy data.

While large brands often have a wide horizontal spread of the products they offer, small to medium sized brands are often tightly centered in a specific market sector. By examining the taxonomic distribution of such a brand's products, we can discover anomalous products which are potentially misbranded or miscategorized.

Anomaly detection can be greatly empowered through the use of contextual information attributed to brands and taxonomies. Such data, such as an association between a brand and the type of products they are associated with producing, can be mined from external sources such as Wikipedia, Wikidata, or news articles. By mining such attributes and materializing them in the KG we have additional signals that allow us to find anomalies. For example, in the KG we can assert that

- (Sprite, ownedBy, The Coca-Cola Company),
- (Sprite, produces, lemon-lime beverages),
- (Coca-Cola, ownedBy, The Coca-Cola Company),
- (Coca-Cola, produces, cola beverages).

From this we infer that Sprite and Coca-Cola are owned by the same parent company. We can then query for a Sprite product which is a cola beverage, or a Coca-Cola product which is a lemon-lime beverage. Such a product probably has an incorrect branding or taxonomic categorization, and a likely solution is to either transpose the brand or the taxonomy category for that product.

3.5 Provenance Based

Provenance based reliability scoring examines metadata in combination with the data itself. Common techniques involve examining the *sources* of a particular piece of data. For example, often KG seed data involves information generated through machine learning techniques such as attribute extraction. Whether or not such data is used for external facing purposes, without having a human

in the loop, can depend on a variety of factors such as regulatory compliance.

Testing provenance is additionally important when data is collected from external sources. Crowdsourced databases, such as Wikidata, can be fused into internal knowledge bases to provide additional contextualization of entities such as food items. Since such data sources can be publicly modified, one must take care when the data is used. See [9] for more details regarding this.

As an example of how we use this, suppose k sources $S_1, \ldots, S_k$ make claims about the weight of a product P. We then have k putative triples, $(P, weight, w_k)$. For each potential weight w we can find the fraction of sources which claim w is the true weight:

$$p_w = \frac{\#\{S_k : w_k = w\}}{k}.$$

According to the wisdom of the crowd, a weight w which maximizes p_w is a likely candidate for the true weight (especially as the number of sources grows). However, if this maximal p_w is not close to 1, this indicates disagreement across sources. In particular this may be seen as a sign of unreliability of the underlying fact, and may be cause for a human to step in and see what is causing the confusion.

4 IMPACT ANALYSIS

In this section we will briefly discuss a way of evaluating KG cleaning efforts. The goal will be to display how the precision of a machine learning model can be impacted by the cleanliness of the data on which it was trained. The hypothetical use case will be to develop a label propagation technique which predicts values for items which are missing attributes in the KG. This particular use case is somewhat arbitrary; similar themes could have been discussed with other tasks, such as classification.

Our approach will be to run two regression tasks which are identical except for one fact: the first regression will be trained on all of the relevant knowledge in the *uncleaned* staging KG, whereas the second regression will only be trained on the high quality data within the KG. We will see that the second regression has a *lower* root mean squared error (by roughly 4%), showing that cleaner data sets can improve downstream tasks related to machine learning.

The experimental framework is as follows. We have sampled 300,000 products from the KG which have nutritional information for proteins, carbohydrates, and fats per serving. The goal will be to train a simple model which predicts the amount of protein per serving in a product, based on that product's carbohydrates and fats per serving, together with dummy variables based on taxonomy:

$$fats_i = f(carbs_i, proteins_i, taxonomy_i).$$

As we are more concerned with the impacts regarding cleanliness of the underlying training data, rather than the model itself, we will take f to be a simple decision tree regressor.

Of the 300,000 sampled products, 3,164 of them failed one or more unit tests related to their carbohydrate, protein, or fat information. Based on these tests, we split the data into 3,164 "noisy" products and 296,836 "clean" products. As the clean products are representative of "true" data, we will randomly sample and set aside an evaluation set of 20% of the clean products. Note that *the same evaluation set* will be used for both regressions. This gives the following distribution of training points across the two regressions:

	Training Set Size	Percent Clean	Percent Noisy
R1	237,469	100%	0%
R2	240,633	98.69%	1.31%

Table 1: Training setup for Regression 1 and Regression 2; all other factors are held constant.

With this setup we used SciKit Learn to train a decision tree regressor and evaluated both models on the testing set. Again, as our emphasis is solely on the impact of the underlying training data quality, we simply used the default SciKit Learn regressor with no hyperparameter tuning. The mean squared error and root mean squared error for both regressors are outlined below:

	MSE	RMSE	R^2
R1	14.00	3.74	.5914
R2	15.37	3.92	.5514

Table 2: Model evaluation for both regressions, displaying a 4.6% drop in RMSE from Regression 2 to Regression 1.

While the absolute difference between the errors above is small, we emphasize that the difference reflects a 4.6% drop in RMSE when moving from training on unclean data vs training on clean data.

The test above hints at the efficacy of focusing on data cleaning and reliability. The causes of this lowering in validation error were a straightforward outlier detection scheme combined with seven simple constraint based tests regarding nutritional information. We believe this serves as a simple proof of concept and that more refined KG cleaning techniques would lead to even greater impacts.

5 CONCLUSION AND NEXT STEPS

In this paper we have given an overview of an initial reliability system that can be used to score data in a KG based on its quality. This system is widely applicable, and can be implemented in knowledge graphs of any variety. These techniques can provide safeguards against bad data from corrupting downstream tasks, such as product retrieval during search. They also help in providing clean data sets for ML training and in targeting costly data validation efforts into areas of the catalog that need it most. The simplicity of our tests in comparison to techniques such as [2] allowed us to obtain tangible benefits quickly and with minimal technical overhead.

As a concrete way to measure the impact of this framework we ran two regressions to predict the amount of protein in foods; the first regression was trained on all of the relevant data in the KG, whereas the second was only trained on the highest quality data. The 4.6% drop in RMSE is taken to be a positive signal that KG reliability scoring is a worthwhile endeavor.

Looking forward, we hope to refine and build upon this reliability system. The existence of reliability scores provides a starting point for a static ranking of Instacart products, which will allow us to recall products with complete, high quality information regarding them during search. This will lead to a better customer experience. These goals will benefit from a continually growing set of reliability measures, including ML techniques such as link prediction, which can be built on top of our current system.

REFERENCES

[1] Daniel Fleischhacker, Heiko Paulheim, Volha Bryl, Johanna Völker, and Christian Bizer. 2014. Detecting Errors in Numerical Linked Data Using Cross-Checked Outlier Detection. In *ISWC '14 Proceedings of the 13th International Semantic Web Conference - Part I*. 357–372.

[2] Congcong Ge, Yunjun Gao, Honghui Weng, Chong Zhang, Xiaoye Miao, and Baihua Zheng. 2020. KGClean: An Embedding Powered Knowledge Graph Cleaning Framework. *arXiv preprint arXiv:2004.14478* (2020).

[3] Mohamed Gharibi, Arun Zachariah, and Praveen Rao. 2020. FoodKG: A Tool to Enrich Knowledge Graphs Using Machine Learning Techniques. *Frontiers in Big Data* 3 (2020), 12–12.

[4] Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard de Melo, Claudio Gutierrez, José Emilio Labra Gayo, Sabrina Kirrane, Sebastian Neumaier, Axel Polleres, Roberto Navigli, Axel-Cyrille Ngonga Ngomo, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. 2020. Knowledge Graphs. *arXiv: Artificial Intelligence* (2020).

[5] Natasha Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. 2019. Industry-scale knowledge graphs: lessons and challenges. *Communications of The ACM* 17, 2 (2019), 20.

[6] Stefano Ortona, Venkata Vamsikrishna Meduri, and Paolo Papotti. 2018. RuDiK: rule discovery in knowledge bases. In *Proceedings of the VLDB Endowment*, Vol. 11. 1946–1949.

[7] Heiko Paulheim. 2016. Knowledge graph refinement: A survey of approaches and evaluation methods. *Social Work* 8, 3 (2016), 489–508.

[8] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 1532–1543.

[9] Alessandro Piscopo, Lucie-Aimée Kaffee, Chris Phethean, and Elena Simperl. 2017. Provenance information in a collaborative knowledge graph: an evaluation of Wikidata external references. In *International Semantic Web Conference (1)*. 542–558.

[10] Robert Speer, Joshua Chin, and Catherine Havasi. 2017. ConceptNet 5.5: An Open Multilingual Graph of General Knowledge. In *Thirty-First AAAI Conference on Artificial Intelligence*. 4444–4451.

[11] Gerhard Weikum, Xin Luna Dong, Simon Razniewski, and Fabian Suchanek. 2021. Machine Knowledge: Creation and Curation of Comprehensive Knowledge Bases. *Foundations and Trends® in Databases* 10, 2-4 (2021), 108–490. https://doi.org/10.1561/1900000064

[12] Dominik Wienand and Heiko Paulheim. 2014. Detecting Incorrect Numerical Data in DBpedia. In *The Semantic Web: Trends and Challenges*, Valentina Presutti, Claudia d'Amato, Fabien Gandon, Mathieu d'Aquin, Steffen Staab, and Anna Tordai (Eds.). Springer International Publishing, Cham, 504–518.