A Flexible Algorithmic Approach for Identifying Conflicting/Deviating Data on the Web

Nour Jnoub, Wolfgang Klas, Peter Kalchgruber, Elaheh Momeni
Multimedia Information System
Faculty of Computer Science
University of Vienna
Währinger Str. 29, 1090 Vienna, Austria
firstname.lastname@univie.ac.at

***Abstract*—Information on the Web often contains contradictions and conflicting information, thus impacting the quality of data sources and the quality-related performance of search and retrieval. Therefore, appropriate techniques need to be developed and integrated into the infrastructure serving for the retrieval and browsing of data sources such that conflicting data are detected, can be removed or blocked, or can be highlighted to the user in order to offer an improvement of the quality of content consumed by users. This paper proposes an approach which allows to detect conflicting data by providing a technique for investigating deviation between values available from structured data on the Web. Our approach consists of multiple phases: First, some initial pre-processing of data from targeted data sources prepares the data sources to be comparable. Second, Levenshtein distance is computed between data elements to represent the degree of conflict between data elements. Third, computing the cosine similarity between vectors of Levenshtein distance values and a user-configurable sensitivity vector, encoding the characteristics of a specific kind of conflict that is subject to investigation, finally allows for a ranked detection of the conflicting data. This algorithm has been applied and tested on a data collection about movies from the Web, illustrating how the techniques can be applied for the detection of conflicting information on the Web.**

***Index Terms*—Conflicting Data, Levenshtein distance, Cosine similarity**

I. Introduction

Data sources have become accessible worldwide by almost any kind of information system built using Web infrastructure and Web technology. Obviously, such external data sources may be combined and integrated, becoming a central part of information systems offered to users. A very impressive example for almost unlimited availability and highly interconnected data is the Linked (Open) Data Cloud [1], which is a set of best practices for publishing and connecting structured data on the Web. Another important initiative is Schema.org [2], which is a collaborative, community activity with a mission to create, maintain, and promote schemas for structured data on the Internet, on web pages, in email messages, and beyond. Structured data embedded into data sources available on the Web on the one hand is becoming increasingly important for building information systems based on a variety of data sources, on the other hand many of the big industrial IT-players like Google and Microsoft are pushing the embedding and usage of structured data in data sources for the Web by exploiting structured data to improve their products and services.

Those data sources might provide overlapping, but quite often inconsistent information, because of incomplete information, errors, different abstraction levels of the data, etc. In order to maintain a high quality of information consumption experience to the user it is important to at least detect such inconsistencies or conflicting information, to inform the user about such conflicting information in an appropriate way, and maybe to provide a proposal for resolving the inconsistencies in order to increase the quality of information sources for the benefit of the information owners as well as the information consumers (users or machines). FactCheck [3] is an approach which aims at offering such services integrated into a novel Web infrastructure. A very central component of the FactCheck services is the detection and possible analyzing of inconsistencies or conflicting information on the Web.

This paper is presenting a specific approach to identify conflicting data on the web, making use of embedded structured data on the Web. We present an algorithm which is simple but effective for the investigation and detection of some specific kind of conflicting or deviating data on the web.

The detection algorithm uses (a) Levenshtein distance, (b) cosine similarity, and (c) a novel concept of control parameter which we call sensitivity vectors.

A sensitivity vector allows for specifying a specific kind of deviation of data values that one would like to investigate further in the dataset in order to detect conflicting data. By applying different sensitivity vectors one can analyze the dataset with respect to various kinds of deviations of data values compared to some reference data element.

The remainder of this paper is organized as follows: Section II briefly outlines a set of related works. In section III we present the proposed approach. In section IV we cover experimental results to illustrate the usage of the algorithm. Section V concludes the paper and gives an overview of future work.

II. Related Works

Recently, different approaches have been integrated and developed from database community for resolving the issue of conflicting records, especially when integrated from several sources.

Approaches for dealing with conflicted records in data integration [4], [5], [6] and [7] have a common issue, that they all depend on majority voting. This means that the records with high number of votes will be considered as valid return answers. But the problem of such a system is that despite the fact that the votes are from diverse sources, they have been equally weighted. These systems do not consider various levels of reliability of sources, which very often does not reflect the reality. Furthermore, estimating the origin reliability to detect the correct data and avoid conflicts is hard to achieve, mainly when one or more of the sources keep emitting low-quality data.

Moreover, beginning from the significant role of source reliability, many fact detection approaches have been developed to indicate the correct data without supervision [8], [9], and [10]. Furthermore, [11] and [12] proposed different algorithms that lead to different levels of complexity based on probabilistic models. Additionally, Kasneci et al. [13], introduced "wisdom of the crowds" by aggregating truth assessments that users feeds regarding statements. Thus, the CoBayes is a framework which uses a probabilistic model of the truth values of statements and the expertise of users for assessing statements.

Zhao et al. [14] suggested a probabilistic graphical model that can automatically infer true records and source quality without any supervision.

However, the previous systems are developed specially for a single type of data; thus, heterogeneous types of data are not well considered. In real life scenarios, heterogeneous data predominantly exists. For example, in health care databases, when combining multiple records, a patients record might contain different heterogeneous data sources. In contrast to that, our proposed approach (a) is not related to any kind of textual data, (b) does not require a probabilistic or a statistical model whose parameters should be estimated for each scenario.

III. Approach

Given a selected data record from a collection of data records as a reference item our approach allows for investigating different kinds of deviations between the values of the reference item and the values of the data elements in the dataset.

The proposed approach consists of three phases.

First, preprocessing the data including cleaning, unifying, etc. in order to make data elements comparable may be needed. For example, these preprocessing steps may include the following:

- For attributes of type date all dates should be converted to a unified date format, e.g., day/month/year or year-month-day.
- For attributes of type string only characters from a well-defined alphabet (e.g., converted to unified lower or upper case) should be kept using, e.g., regular expressions.
- For numeric attributes only numeric values may be kept and well structured using, e.g., regular expressions.

The second phase is about computing the pairwise deviations between the values in the data records of the dataset and the values in the reference data record by means of normalized Levenshtein Distance. The resulting matrix of Levenshtein distance values will hence encode the extensiveness of, e.g., misspellings in string values or the deviation of actors names. Obviously, as Levenshtein distance counts the editing steps needed to transform a value to another value, the notion of deviation is very much a syntactic one.

The third phase is based on the resulting matrix of Levenshtein distance values. It is about the computing of a ranked deviation list which can be controlled by a so-called sensitivity vector. The sensitivity vector allows specifying a specific kind of deviation that one would like to investigate in the dataset. For example, one might be interested to find the most deviating records compared to the reference item, or to find records showing some very specific deviation in some parts of the record, or one might be interested to find the non-deviating records, etc..

In the following, we will present the algorithm for phase two and three.

A. Computing deviation between data records

Algorithm 1 shows the pseudo code for computing the pairwise deviation between the reference item and the individual records in the dataset. The algorithm needs the following input parameters:

- $Dataset$ denotes the globally available dataset encoded as a table.
- $RefItem$ denotes the data record consisting of attribute-value pairs, that serves as the reference item all the other records in the dataset are to be compared with.

The algorithm checks the similarity between the $RefItem$ and each record in the $Dataset$ using Levenshtein distance (see Algorithm 1, line 4). Levenshtein distance is a string metrics for measuring the difference between two character sequences. It indicates the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string to the other [15].

The Levenshtein distance is then normalized by dividing each Levenshtein distance value by the length of the longest string of the compared attributes within the pairwise similarity calculation (see Algorithm 1, line 5 and 6). The normalization equation used is shown in Equation (1),

$$NormalizedLevenDist(str1, str2) = \frac{LevenDist(str1, str2)}{max(length(str1), length(str2))} \quad (1)$$

where $LevenDist$ is the Levenshtein distance function, $str1$ and $str2$ are the strings to be compared, max computes the maximum of the $length$ of the two given strings.

The result of computed similarities are then returned as normalized Levenshtein distance matrix.

Algorithm 1 CompDev

```
Input:
    global $Dataset$ : table with M data tuples, each consisting of D elements
    $RefItem$ : vector consisting of D attribute-value pairs
Output:
    $norm\_LD\_matrix$ : Levenshtein Distance Matrix (of dimension M x D)
1:  function CompDev($RefItem$)
2:      for $i = 1$ to $M$ do
3:          for $j = 1$ to $D$ do
4:              $ld \leftarrow LevDistance(RefItem[j].value, Dataset[i, j])$
5:              $maxLen \leftarrow max(length(RefItem[j].value), length(Dataset[i, j]))$
6:              $norm\_LD\_matrix[i, j] \leftarrow ld/maxLen$
7:          end for
8:      end for
9:      return $norm\_LD\_matrix$
10: end function
```

B. Computing ranked deviation list

Algorithm 2 shows the pseudo code for computing the cosine similarity between sensitivity vector $SenVec$ and the normalized Levenshtein distance matrix $LDmatrix$, resulting from the previous computing step. The sensitivity vector $SenVec$ denotes a control parameter encoding a specific degree of deviation of data values that one is interested to investigate.

A sensitivity vector is of the same dimension D as the $RefItem$, the reference record used in Algorithm 1. The set of possible sensitivity vectors is $Value^D$ with $Value \in [0, 1]$, because the Levenshtein distance values from Algorithm 1 we use for comparison are between 0 and 1.

Algorithm 2, line 4 computes the Cosine Similarity between $SenVec$ and each row in $LDmatrix$, iterating over all rows, constructing an intermediate matrix $DeviationMatrix$ consisting of the index of a data record and its cosine similarity value. The Cosine Similarity function used is given by Equation (2):

$$CosineSimilarity(SenVec, LevenDist[i]) = \frac{SenVec \times LevenDist[i]}{|SenVec| \times |LevenDist[i]|} \tag{2}$$

where $SenVec$ is a non-zero length sensitivity vector consisting of D values $v \in [0, 1]$, and $LevenDist[i]$ is a non-zero length vector consisting of normalized Levenshtein distance values $v \in [0, 1]$ for a data element i.

In the special case that the record in the dataset does not show any deviation with respect to the reference item, i.e., $|LDmatrix[i]| = 0$, the corresponding similarity value in the $DeviationMatrix$ is set to 0 (see Algorithm 2, line 5 and 6).

In the special case that the sensitivity vector has been chosen to express that we are interested in the exact or most similar records (no or almost no deviation with respect to the reference item), i.e., a vector with only 0 values, $|SenVec| = 0$, the corresponding similarity value in the $DeviationMatrix$ is set to a value depending on the average deviation encoded for the D attributes of the data record (see Algorithm 2, line 7 and 8).

Finally, the constructed $DeviationMatrix$ is sorted by the similarity values computed before in descending order, which creates the final ranked list of records to be returned.

Algorithm 2 CompRankDevVecs

```
Input:
    $LDmatrix$ : Levenshtein Distance Matrix (of dimension M x D)
    $SenVec$ : sensitivity deviation vector (of dimension D)
Output:
    $RankedDevMatrix$ : ranked deviation of data filtered by sensitivity vector
1:  function CompRankDevVecs($LDmatrix$, $SenVec$)
2:      for $i = 1$ to $M$ do
3:          if $|SenVec| \neq 0$ and $|LDmatrix[i]| \neq 0$ then
4:              $DeviationMatrix[i] \leftarrow (i, CosineSimilarity(SenVec, LDmatrix[i]))$
5:          else if $|SenVec| \neq 0$ and $|LDmatrix[i]| = 0$ then
6:              $DeviationMatrix[i] \leftarrow (i, 0)$        ▷ as there is no deviation in data
7:          else if $|SenVec| = 0$ then
8:              $DeviationMatrix[i] \leftarrow (i, 1 - \frac{1}{D} \times \sum_{j=1}^{D} LDmatrix[i, j])$
9:          end if
10:     end for
11:     $RankedDevMatrix \leftarrow$ sort $DeviationMatrix$ according to similarity values in the 2nd column, descending.
12:     return $RankedDevMatrix$
13: end function
```

C. Application scenarios of investigating data deviation

Let $Dataset$ be a collection of movie descriptions, consisting of movie title, release date, the name of the movie director. Let us assume $Dataset$ has been preprocessed according to phase 1 described above.

Let $RefItem$ be a description of a single movie subject to our interest of investigating the data collection, e.g., it might be a movie description known to be perfectly correct. E.g., $RefItem = ("Title" = "Flubber", "ReleaseDate" = "1997-11-15", "Director" = "LesMayfield")$.

Scenario 1: Looking for significantly conflicting data

Looking for those movies that show high deviation in their data compared to RefItem, ranked from highest to lowest, can be computed by choosing a sensitivity vector SenVec = (1, 1, 1), i.e., $CompRankDevVecs(CompDev(RefItem), (1, 1, 1))$.

This results in a ranked list of movies containing movies with a completely different title, release date, and director name on top of the list, and movies with the identical/same title, release date, and name of director at the end of the ranked list.

Scenario 2: Looking for non-conflicting data

Looking for those movies that show no or lowest deviation in their data compared to RefItem, ranked from non-deviation to high-deviation, can be computed by choosing a sensitivity vector SenVec = (0, 0, 0), i.e., $CompRankDevVecs(CompDev(RefItem), (0, 0, 0))$.

This results in a ranked list of movie descriptions containing movies with exactly the same title, release date, and director name on top of the list, and movie descriptions with the diverging title, release date, and name of director at the end of the ranked list.

Scenario 3: Looking for partially conflicting data (1)

Looking for movies that show highest deviation in the movie title (first attribute), but no or lowest deviation in their other attributes compared to RefItem, ranked from non-deviation to high-deviation, can be computed by choosing a sensitivity vector SenVec = (1, 0, 0), i.e., $CompRankDevVecs(CompDev(RefItem), (1, 0, 0))$.

This results in a ranked list of movies containing movies with a completely different title, but similar release date and director values on top of the list, and movies with the identical/same title at the end of the ranked list.

Scenario 4: Looking for partially conflicting data (2)

Looking for movies that show highest deviation in the movie title (first attribute) and the name of the director, but no or lowest deviation in the release date compared to RefItem, ranked from non-deviation to high-deviation, can be computed by choosing a sensitivity vector SenVec = (1, 0, 1), i.e., $CompRankDevVecs(CompDev(RefItem), (1, 0, 1))$.

The outcome results is a list of movies ranked from the most conflicted movies on the top of the list to the identical ones in the bottom of the list.

IV. Experimental Results

In this section we want to demonstrate the performance of the proposed approach. The prototype is implemented in Python to gain sufficient information and prove the applicability of our approach.

A. Dataset

The dataset is extracted from the website IMDb[1]. The data includes different conflicting descriptions for movies. For example, the $Title$ attribute may contain different additional unknown values such as "25th Hour (2002)" and "25th Hour". Similarly, the $ReleaseDate$ attribute may contain different entries; e.g., "24 March 2008" and "2008.03.24". Additionally, regarding the $Actors$ attribute, some actors may appear for a movie, and another record of the same movie may not contain the same list of actors.

We extracted 500 records (movies). Each record included attributes $Title$, $ReleaseDate$, and $Actors$. The reason of choosing 500 movies is due to the fact that the considered movies are used as a validation (test) dataset, since the exact number of data conflicts in movies is known. The 500 movies have been checked carefully. Thus, we can investigate the performance of the proposed conflict detection algorithm. The dataset contains 346 unique movies. The rest, i.e., 154 movie descriptions, can be considered as conflicting ones, deviating from the unique movies, or - as a special case - are exact duplicates of one of the unique movies.

B. Results

In the experiment we present here we have chosen 5 sample movies to serve as reference records ($RefItem$). In order to investigate different kinds of deviations of movie descriptions present in the dataset, we applied several sensitivity vectors encoding these different kinds of deviations[2].

Table I shows examples of the number of top ranked conflicting movies from the validation dataset for the five sample movies serving as reference items. All these reference records (i.e., $RefItem$) consist of three attributes, $Title$, $ReleaseDate$, and $Actors$.

For the illustrated examples, the result for the sensitivity vector $(0, 0, 1)$ for a given movie as $RefItem$ tells us that all the existing conflicting movies in the dataset (see last column) show deviating values for the third attribute, i.e., the list of actors. E.g., when comparing to the reference movie "Going on", all the 9 movie descriptions in the dataset, which show conflicting values in the list of actors, but show the same values for $Title$ and $ReleaseDate$, are ranked on the top 9 positions by our algorithm. In this case the algorithm shows excellent accuracy.

The results for the sensitivity vector $(0, 1, 1)$ tell us that for the first two reference movies in the table all the existing conflicting movie descriptions in the dataset (see last column) show deviating values for both, the second ($ReleaseDate$) and third ($Actors$) attributes and all are ranked on the top of the list. For the last reference movie in the table only one of the four conflicting movie descriptions was top ranked. The other three movie descriptions were not positioned in the top four positions, but ranked lower or handled as exact duplicates of the reference movie and not received as conflicting elements. Table I also illustrates the variety of deviations one can investigate, as the variations of sensitivity vectors shown in the example encode different kinds of deviations in the movie descriptions.

V. Conclusion

When trying to detect conflicting descriptions for the same information artefacts on the web, one needs to detect deviations between data elements describing those

[1]http://www.imdb.com/title/tt0119137/

[2]Because the experiment does not look for non-conflicting data elements, the sensitivity vector $(0, 0, 0)$ is not included in the table.

TABLE I
NUMBER OF TOP-RANKED CONFLICTING MOVIES FOR 5 SAMPLE MOVIES APPLYING DIFFERENT SENSITIVITY VECTORS WITH RESPECT TO ATTRIBUTES $Title, ReleaseDate, Actors$

Reference Movie/SenVec	**[0 0 1]**	**[0 1 0]**	**[1 0 0]**	**[1 0 1]**	**[1 1 0]**	**[0 1 1]**	**[1 1 1]**	**# of real conflicting movies**
5th Hour	6	2	2	2	2	6	2	6
Jump Street	7	1	1	1	1	7	1	7
A Space Odyssey	7	3	3	3	3	4	3	7
Going on	9	2	2	1	2	7	1	9
Til We Meet Again	4	1	1	1	1	1	1	4

artefacts. We presented an approach that allows to compare structured data elements and investigate them with respect to deviations in their attribute values. The proposed approach consists of three phases: (i) an initial preprocessing phase aiming at syntactically cleaning data to make it comparable, (ii) a comparison phase computing the pairwise deviations between values of the data elements and a reference data element subject to further investigation. The deviation is measured based on Levenshtein distance. (iii) Depending on a given sensitivity vector, which encodes a specific kind of deviation one is interested to investigate, the third phase computes a ranked list of deviations in a dataset compared to a given reference item.

The concept of sensitivity vectors serves as a kind of parameter for the approach, as sensitivity vectors control the way how deviations between data elements are to be taken into account for the assessment of potentially conflicting data elements. As sensitivity vectors can encode almost any combination of deviations one might be interested in when detecting conflicting data elements, the proposed approach becomes a very powerful tool for the analysis of conflicts of data elements.

We illustrated the application of the proposed algorithm by means of an experiment, a collection of movie descriptions available on the Web. For given movie descriptions we can detect conflicting descriptions about the same movie, considering various deviations, encoded by various sensitivity vectors. The experiments proof the flexibility of the approach as a tool to investigate various kinds of conflicting data on the Web. An interesting advantage of the algorithm is not to require any training phase or parameter estimation of any probabilistic or statistical model.

Moreover, it should be mentioned that, although there are numerous machine learning approaches that can be applied for fact check purposes, applying such techniques still have some disadvantaged [16]. For example, for supervised learning techniques accumulating a sufficient training set presents a challenge in our application. On the other hand, unsupervised learning techniques have advantages, e.g., they can recognize new classes that may not have been considered before. Therefore, this can be done without explicit training that leads to the advantage of unknown classes or unusual classes. Extending the algorithm by means of unsupervised learning techniques will allow the fact check detection system perform also in a more autonomous manner.

The algorithm presented will be integrated with the FactCheck framework [3] which aims at providing tools for (a) notifying users about conflicting data on Web pages that are visited, (b) offering users four lists of equal, conflicting, locally missing and remotely missing facts, and (c) triggering a notification to data source owners about conflicting or missing data on their web pages.

REFERENCES

[1] "The linking open data cloud diagram," 2018, http://lod-cloud.net.
[2] "Schema.org," 2018, http://schema.org.
[3] P. Kalchgruber, W. Klas, N. Jnoub, and E. Momeni, "Factcheck - identify and fix conflicting data on the web," in *18th International Conference on Web Engineering*. Springer, 2018, p. accepted for publication.
[4] Z. Jiang, "A decision-theoretic framework for numerical attribute value reconciliation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 7, pp. 1153–1169, 2012.
[5] J. Bleiholder and F. Naumann, "Conflict handling strategies in an integrated information system," 2006.
[6] X. L. Dong and F. Naumann, "Data fusion: resolving data conflicts for integration," *Proceedings of the VLDB Endowment*, vol. 2, no. 2, pp. 1654–1655, 2009.
[7] J. Bleiholder and F. Naumann, "Data fusion," *ACM Computing Surveys (CSUR)*, vol. 41, no. 1, p. 1, 2009.
[8] L. Blanco, V. Crescenzi, P. Merialdo, and P. Papotti, "Probabilistic models to reconcile complex data from inaccurate data sources," in *International Conference on Advanced Information Systems Engineering*. Springer, 2010, pp. 83–97.
[9] C. Dai, D. Lin, E. Bertino, and M. Kantarcioglu, "An approach to evaluate data trustworthiness based on data provenance." *Secure Data Management*, vol. 5159, pp. 82–98, 2008.
[10] C.-T. Chu, S. K. Kim, Y.-A. Lin, Y. Yu, G. Bradski, K. Olukotun, and A. Y. Ng, "Map-reduce for machine learning on multicore," in *Advances in neural information processing systems*, 2007, pp. 281–288.
[11] A. Galland, S. Abiteboul, A. Marian, and P. Senellart, "Corroborating information from disagreeing views," in *Proceedings of the third ACM international conference on Web search and data mining*. ACM, 2010, pp. 131–140.
[12] J. Pasternack and D. Roth, "Making better informed trust decisions with generalized fact-findingqi2013mining," in *IJCAI*, vol. 11, 2011, pp. 2324–2329.
[13] G. Kasneci, J. V. Gael, D. Stern, and T. Graepel, "Cobayes: bayesian knowledge corroboration with assessors of unknown areas of expertise," in *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM, 2011, pp. 465–474.
[14] B. Zhao, B. I. Rubinstein, J. Gemmell, and J. Han, "A bayesian approach to discovering truth from conflicting sources for data integration," *Proceedings of the VLDB Endowment*, vol. 5, no. 6, pp. 550–561, 2012.
[15] L. V, "Binary codes capable of correcting deletions, insertions, and reversals," *Cybernetics and Control Theory*, vol. 10, no. 8, pp. 707–710, 1966.
[16] D. Isaac and C. Lynnes, "Automated data quality assessment in the intelligent archive," *White Paper prepared for the Intelligent Data Understanding program*, vol. 17, 2003.