DATA SCIENCE JOURNAL

Liao, X, et al. 2019. A Column Styled Composable Schema Matcher for Semantic Data-Types. *Data Science Journal,* 18: 25, pp. 1–15. DOI: https://doi.org/10.5334/dsj-2019-025

RESEARCH PAPER

# A Column Styled Composable Schema Matcher for Semantic Data-Types

Xiaofeng Liao, Jordy Bottelier and Zhiming Zhao

System and Network Engineering Lab, Informatics Institute, University of Amsterdam, Amsterdam, NL

Corresponding author: Zhiming Zhao (editor@codata.org)

Schema matching exists as a long-standing challenge in many database related applications, such as data integration, where two databases with different schema have to be integrated. With the evolvement from database to big data, the schema matching has been enriched with various purposes and application contexts, ranging from data integration, to service integration, to semantic data clouding, until more recent exploratory data analysis over big data. These enriched contexts increase the demand for schema matching between semantic data-types, such as XML, RDF etc.

The existing integration approaches have not dealt with the challenges of defining a relation between XML and other semantic data-types. To address these challenges, this paper studies the problem of schema mapping from XML to RDF in two folds. Firstly, testify the validity of single matcher in a column based manner for the semantic data types. Secondly, testify the validity of a highly configurable framework that utilizes hierarchical classification in order to construct a composable pipeline.

We propose and implement a Reconfigurable pipeline for Semi-Automatic Schema Matching (REPSASM), which aims to solve the customizability of the matching problem by providing an environment in which a user can create, configure and experiment with their own schema-matching procedure.

The experiments performed within this work show that the configurability and hierarchical classification improves the matching result, and it proposes an algorithm to automatically optimize such a hierarchy pipeline.

**Keywords:** Schema Matching; Semantic Data-types; XML; RDF

## I. Introduction

Schema matching exists as a principle problem in many database related applications, such as data integration, where two databases with different schema have to be integrated. It is described as the task of identifying semantically equivalent or similar elements in two different schema (Rahm and Bernstein, 2001).

With the development from database to big data, the schema matching problem has been enriched with various purposes and application contexts, ranging from data integration, to service integration, to semantic data clouding, until more recent exploratory data analysis over big data. These enriched contexts increase the demand for schema matching between semantic data-types, such as XML, RDF etc.

As XML schema do not express semantics but rather the document structure, there is a lack of semantic interoperability regarding current (XML-based) metadata standards. By using Semantic Web technologies, ontologies can be created to describe the semantics of a particular metadata format. A problem is that the existing XML data (compliant with a particular meta-data format) cannot be used by an ontology, implying the need for a conversion of XML data to RDF instances (Davy Van et al., 2008).

Currently, schema matching is typically performed manually. Obviously, manually specifying schema matches is a tedious, time-consuming, error-prone, and therefore expensive process.

This is a growing problem given the rapidly increasing number of web data sources and E-businesses to integrate.

In this situation, automated data integration offers opportunities to solve these problems by letting machines interpret the data and automatically create a mapping based on semantic or syntactic features.

Automating part of or the entire processes of schema mapping can essentially accelerate the data integration procedure of human experts and thus reduce the overall time cost. However, several challenges make such automated mapping difficult or even impossible. Many problems can occur during the mapping process. Matches might not be found, or even worse, false positives are found. In addition, one data source might not fully match with the other data source, data source A could contain information that does not cohere with the data found in data source B. Source A could also contain less information than source B, in which case a complete mapping is impossible.

Since completely automating the mapping process could be impossible in certain cases, human interpretation can not be excluded from the mapping process. This is why this work focuses on semi-automating the process, which could reduce the time cost for creating a mapping.

A semi-automated solution could be that a system predicts the mapping based on the schema contents, a human expert could then evaluate the predicted mapping. There are many challenges and questions that need answering when creating such a system.

This study will aim to answer the following question:

- Whether the column mannered schema matching method can also be applied to non-column type data structures, such as XML to RDF?
- How can an effective semi-automated schema matching pipeline be created and customized?

We propose and implement an reconfigurable pipeline for Semi-Automatic Schema Matching (REPSASM), in this context as a chain of matchers that is used to classify data. The goal of such a pipeline in the schema matching context should be to semi-automatically map new schema into a pre-defined global schema and solve the cusomizability of the matching problem by providing an environment in which a user can creat, configure and experiment with their own schema-matching pipeline. We will refer to this pre-defined global schema as the "target-schema" or simply "target". We will refer to the new schema that need to be mapped as the "source schema" or simply "source".

The remainder of the paper is organized as follows. In Section II, we briefly present the related work. Section III describes the challenges then introduces the design and implementation of the REPSASM system. Section IV presents experiments setup and analyze the results. Finally, Section V concludes this paper.

## II. Related Work

In this section, we present two research directions that are related to our paper: (1) Transforming or mapping XML data to RDF ontology; (2) The optimization of the matching process and the automation effort.

First, there are various approaches related to mapping or transforming XML data into RDF.

XML Schema (Biron et al., 2004) was designed as a language to make XML validation possible with more expressiveness than DTDs (Bex et al., 2004). Using XML Schema, developers can define the structure, constraints and documentation of an XML vocabulary. In the Semantic Web, RDF was missing a standard constraints validation language which covers the same features that XML Schema does for XML. Some alternatives were OWL (McGuinness et al., 2004) and RDF Schema (Brickley et al., 2014), however, they do not cover completely what XML Schema does for XML (Tao et al., 2010).

Some approaches are proposed to measure the element similarity between schema. (Do and Rahm, 2002) computes the name similarity between elements of two XML schema. (Yang and Powers, 2005) uses linguistic taxonomy based on concept definitions in WordNet (Miller, 1995) to gain the most accurate semantics for the element names. Some researchers (Nayak & Tran, 2007, Algergawy et al., 2010) employ supplemental functions to calculate the similarity of a particular feature of a given schema. (Thuy et al., 2012) measures the similarity between XML schema by computing both the structural and semantic aspects of XML elements.

(Thuy et al., 2013) proposes a technique to measure the similarity of duplicate element and based on the similarity results, duplicates are transformed into appropriate RDF concepts.

(Klein, 2002, Thuy et al., 2007) both convert XML data to RDF data by using RDF Schema vocabularies.

(Amann et al., 2001) relies on DTD to define the meaning for every XML element and use XPath to map information in XML documents to ontology.

(Davy Van et al., 2008) presents a generic approach for the transformation of XML data into RDF instances. The transformation is based on the OWL ontology (which corresponds to the metadata format) and an XML document containing rules that describe a mapping between the XML Schema and the OWL ontology.

(Garcia-Gonzalez and Labra-Gayo, 2018) describes a solution to make the conversion from XML Schema to ShEx (Prud'hommeaux et al., 2014), which is proposed to fulfill the requirement of a constraints validation language for RDF.

(Breitling, 2009) presents a technique for making standard transformation between XML and RDF using XSLT.

(Bischof et al., 2012) describes XSPARQL, which is a framework that enables the transformation between XML and RDF based on XQuery and SPARQL and solves the disadvantages of using XSLT for these transformations. However, these works are not covering the schemata mapping problem.

To our best knowledge, this is a first work utilizing column based matcher for XML to RDF matching.

Second, regarding the configurability of the matching process, there are already existing matchers such as Automatch (Berlin and Motro, 2002), COMA (Do and Rahm, 2002), and SemInt (Li and Clifton, 2000). They all focus on different aspects and implementations of schema matching.

COMA (Do and Rahm, 2002) follows a composite approach, which provides an extensible library of different matchers and supports various ways for combining match results. The system utilizes schema information, such as element and structural properties. COMA operates on XML structures and returns matches on an element level.

Automatch (Berlin and Motro, 2002) is a single strategy schema matcher that uses a Naive Bayes matching approach. It uses instance characteristics to match attributes from a relational source schema to a previously constructed global schema.

SemInt (Li and Clifton, 2000) computes a feature vector for each database attribute with values ranging from 0 to 1. Schematic data and instance data are both used in this process. These signatures are then used to first cluster similar attributes from the first schema and then to find the best matching cluster for attributes from the second schema.

The framework in this work is most similar to Automatch. Automatch is a single strategy schema matcher that uses a Naive Bayes matching approach.

Data in REPSASM is also matched to such a global schema. REPSASM however leaves room for implementation, the users can define and test their own strategies.

The current work in the schema matching field however is limited by a single implementation per matcher. REPSASM differs from the presented matchers in the sense that it is not a fully functioning matcher upon initiation. Users can create and experiment with their own schema matching pipeline in order to customize it to fit their specific problem.

## III. The REPSASM framework

### *A. Challenges and Requirements*

The REPSASM framework has been designed bearing two consideration. The first is to be generic in order to support users in trying different approaches to match schema. To be generic, it means that the framework can process various data, including the common csv format, which is in the column based style, and the semantic data types, like XML, RDF, etc.

The second consideration is to experiment with creating and customizing an effective schema matching pipeline for a given dataset to see if customization leads to an improved matching result. In order to do this the framework should therefore be highly flexible in several places:

- The framework should allow a user to pass any structure or element data from a source or target schema to the framework for pre-processing.
- It should be possible within the framework to create a pipeline of matchers which the user can configure.

### *B. Architecture*

In order to satisfy the requirements from the previous section, the architecture has been split into two separate components depicted in **Figures 1** and **2**.

The pre-processing components are called feature builders, and the matching components are called matcher classes. Based on the requirement, we designed functional components which will be discussed in the upcoming sections. Globally, the framework works in a schema matching context using the following steps:

- Data Collection: Collect all the column data from the target schema for the features builders.
- Feature Building: Build feature vectors for the target schema column data.
- Matcher Building: Use the feature vectors to train a matching component.
- Building a pipeline: Build a pipeline of matchers.
- Load the pipeline: Load the pipeline into the schema matcher.

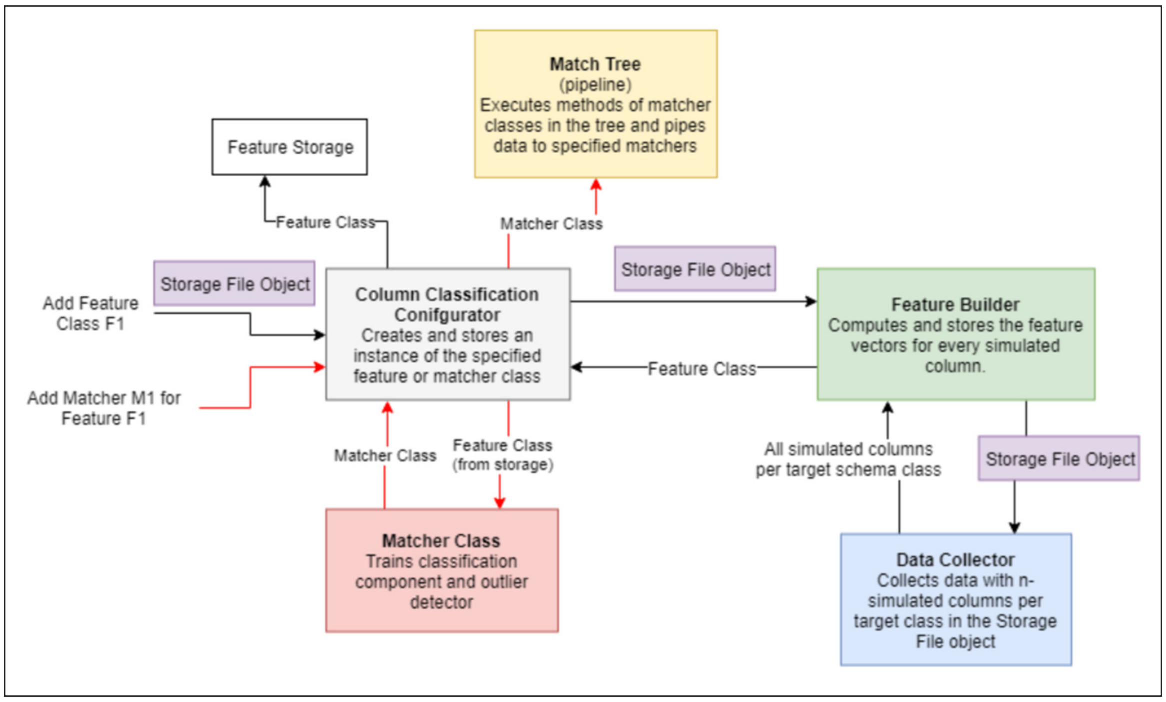
Match Tree
(pipeline)
Executes methods of matcher classes in the tree and pipes data to specified matchers
Feature Storage
Feature Class
Matcher Class
Storage File Object
Add Feature Class F1
Column Classification Conifgurator
Creates and stores an instance of the specified feature or matcher class
Feature Builder
Computes and stores the feature vectors for every simulated column.
Feature Class
Add Matcher M1 for Feature F1
Matcher Class
Feature Class (from storage)
All simulated columns per target schema class
Storage File Object
Matcher Class
Trains classification component and outlier detector
Data Collector
Collects data with n-simulated columns per target class in the Storage File object

**Figure 1:** REPSASM System Architecture.

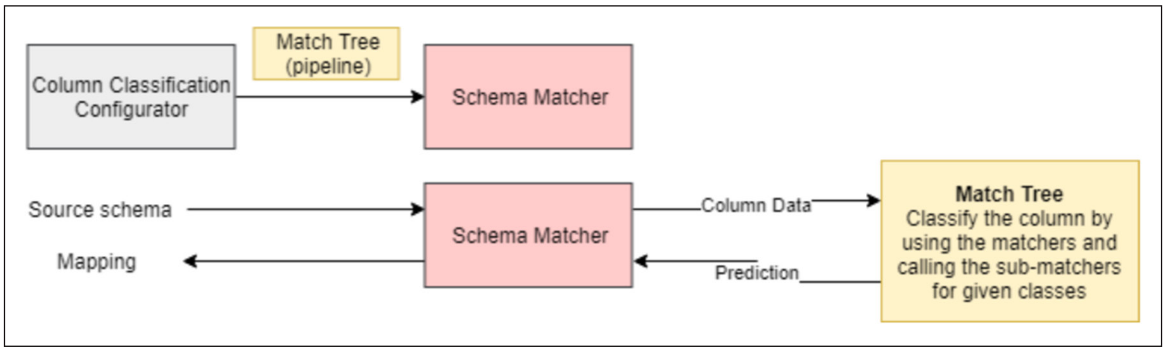
Column Classification Configurator
Match Tree (pipeline)
Schema Matcher
Source schema
Mapping
Schema Matcher
Column Data
Prediction
Match Tree
Classify the column by using the matchers and calling the sub-matchers for given classes

**Figure 2:** REPSASM Schema Matching Architecture.

There are already options for each step created in the framework, but a user is free to implement their own. If they want to build a pipeline of matchers and still be able to use all of the provided methods they should however stick to the given format of methods specified in the framework for each class.

1) *Data Collector:* The first component in the system is the optional data collector. It is optional because the user can also provide data manually. The data collector itself was designed to support an experimentational setup. It has configurable options which allow a user to change one variable within the test setup at a time and then run experiments using the collected data.

2) *Features:* Feature components in the system are designed to further pre-process the target schema data and to store this within the framework for later usage. The data collected by the Data Collector (or ones own data) should be used first by the feature builders. Since this project focuses on learning-based schema matching, the feature builders that have been implemented are used to transform the target schema column data in feature vectors per column upon initialization. The output of the feature builders here is a list of feature vectors and a list of target values (classes). Features should be designed according to the classification needs. Feature builders can be adapted to pre-process any kind of data, and already allow a user to experiment with different algorithms. The feature builders that will be discussed below have been designed to pre-process data for the experiments that will be performed. A user should adapt feature builders according to the data in their target schema.

The goal of a feature is to process the data in such way that a matcher can differentiate between the different columns in the processed data. There are four feature builders already implemented in the framework:

*Fingerprint:* This feature class calculates the datapoints based on the character distributions and n-grams of the inserted column data. For each column, all characters present are counted, as well as the N-grams, and then the result is normalized. N-grams are re-occurring sequences of N characters. They can be used to find patterns in words.

*Syntax Feature Model:* Based on a combinations of the features used in (Nadeau and Sekine, 2007), the syntax feature model is a simple instance-based feature class that checks whether or not the following data is present:

- Instance starts with a capital letter.
- Instance contains multiple words.
- Instance contains multiple capitalized words.
- Instance is all uppercased.
- Instance has special character (x).
- Instance has letter suppercase.
- Instance has letters lowercase.
- Instance has digits.
- Instance is a digit.

*Distributional Representation:* The previously presented features are more syntax based, however, lots of schema contain only textual data. In these cases, a distributional representation generated by a word-embedding model would be more useful. Therefore we already build in a feature which simply creates a corpus out of the instances found in the column. These can later be used by natural language models or word embedding models. The Word2Vec language model (Mikolov et al., 2013) computes the feature vector for every word in a corpus in 200 dimensions. Preliminary experiments determined that these parameters (a uni-gram model with 200 dimensions) would result in the highest accuracy. The Word2Vec model that has already been implemented in REPSASM uses a uni-gram model, this means that every single word is added to the model as opposed to a bi-gram model where only combinations of two words are added.

When training the matchers, the corpus for every column in a class is ran through the model. The feature vectors that are outputted by the model are used as training data for a classifier. Upon classification, a corpus is created from a source-schema column.

*Number Feature:* Lastly, lots of databases contain columns which are solely populated by numbers. These can be hard to separate since their ranges could heavily overlap. Already implemented in REPSASM is a feature class that builds a feature vector based upon the average number in a column, again the character distribution, the average length of the numbers and wether or not it is an integer or a float.

3) *Matchers:* After the features and the targets are computed, they can be utilized by the matchers to classify data. Matchers can consist of rules, or machine learning components. Data from multiple features can be combined by a single matcher or multiple smaller components can classify the data in order to make a prediction. This means these classes can act as a hybrid and as a composite matcher. Matchers can perform differently on different datasets. It would be useful for a user to test their implementation and configuration of a matcher on the initial training data in order to get an indication on how well the matcher can perform on test data. This is why each matcher component can be tested by a test class which implements a k-fold test. During a k-fold test, all the target-schema column data is randomly divided into k equal sized sub-samples. Of the k sub-samples, a single sub-sample is used as the validation data for testing the model, and the remaining sub-samples are used as training data. Such a test can be used to determine if a specific matcher can recognize the given classes with high accuracy. For each feature class there is also an implemented matcher class in the framework. If a user wants to utilize his own matcher class within the entire framework (as opposed to use a single matcher as a stand alone), the user should follow the formats and implementations stated by the framework. These requirements were put on the matcher classes in order for them to all function within the matching pipeline. As long as these requirements are satisfied, any matching algorithm can be implemented. The default classifier (a Support Vector Machine) for each matcher can be overwritten based on the users needs. The machine learning components inside matchers should be trained upon initiation.

4) *Outlier Detection:* Another problem in schema matching is the detection of outliers. Outliers are in this case columns that contain data that is not part of the training set, and therefore does not have to be mapped. If outliers are not recognized but present, they will always be mapped incorrectly. Each class in the framework comes with its own outlier detector. After a matcher classifies data, a specifically trained binary classifier is used to classify the same data. This binary classifier is trained to recognize if data for that particular class is an inlier or an outlier. Outlier data is generated by randomly combining the computed feature vectors from other target-schema classes.

5) *Building Pipelines:* We introduce the hierarchical classification system that uses these classes and allows a user to configure their schema-matching pipeline. It is useful to be able to reproduce such pipeline when iteratively configuring and optimizing it. To accommodate this, the configuration of the entire classification pipeline is stored in the Column Classification Configurator (CCC) object. A user first has to add feature builders to the configuration. After these feature builders are added, a user can add matcher classes which utilize these feature builders. This was done in order to allow a user to easily experiment with different features and matchers. We will first recap on how the implementation of a single matcher works before we discuss how an entire pipeline of matchers can be created.

Upon initiation of the framework there is the training phase in which the features which were added to the CCC collect their data. Data is utilized to train machine learning components which will be utilized to classify columns based on their feature vectors.

After the matching component is trained, this tiny set up can already be used to classify columns.

All columns should obviously be ran through the feature builder and matcher in order to create a mapping for a schema.

When two columns contain syntactical similar information, it is possible that confusion can occur between them during the matching-phase. This can be solved by using a different feature class which could possibly aid in correctly distinguishing the two columns upon classification. Because of this, a user should be able to specify within the framework that a specified matcher should be called when a certain class is predicted. REPSASM allows a user to define pipelines, or 'match trees'. The concept is explained using **Figure 3**.

When a matcher is added to the CCC, it is also inserted into the match tree according to the specification of the user. A user can define if data should be sent to a different matcher depending on the classification result of the previous matcher.

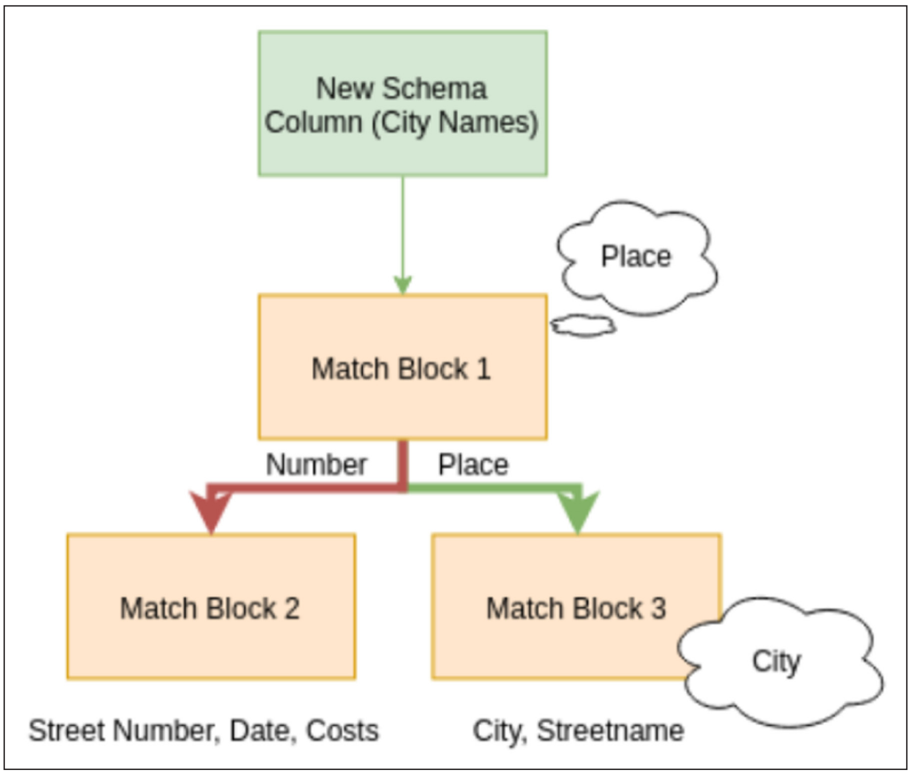

**Figure 3:** Classifying a Column Using the Match Tree.

Matchers are trained upon initialization, before they are added to the entire tree. With the shown code, we first tell the CCC to create an initial matcher which is a Fingerprint matcher. We then add a Number matcher. This matcher is called when the main matcher predicts that the inserted data belongs to the 'number' class. If this is the case, than the data is sent to this matcher for further classification. In **Figure 3**, match block 1 pipes column data to match block 2 in case the column is classified as the 'Number' class. Data is piped to match block 3 if the data is classified as part of the 'Place' class. This setup can be used to classify column data more specifically following the tree-structure.

A pipeline should be fitted specifically for a single global target schema. The outlier detectors can be called in the tree leaves upon classification time to check if a column should be mapped or if it is an outlier. The concept of classifying data in such a tree structure has been inspired by the hierarchical classification field (Silla and Freitas, 2011). Hierarchical classification has not yet been applied in a schema matching context, but could reduce the amount of errors as opposed to a single classifier (Silla and Freitas, 2011, Sun and Lim, 2001).

6) *Loading the Pipeline:* The match tree which the user can create and customize using the CCC can be loaded into the schema matcher component. The schema matching component reads a source schema and calls the methods from the match tree in order to classify column data and create a mapping. By introducing this schema-matching class we separate the mapping of actual schema from the task of building, customizing and using the pipeline. This is done so the match tree can also be used outside of a schema matching context even though this is out of scope for this project.

## IV. Experiments

We introduce experiments to answer the following main-question:

How can an effective semi-automated schema matching pipeline be created and customized for a given dataset?

Therefore we aim our experiments at performance of the previously discussed features and matchers.

We hope to find rough guidelines that users can follow in order to optimize their own schema-matching pipeline.

Since outlier detection is optional and addresses a different kind of matching problem (namely the problem of excluding columns from the mapping), the experiments will be ran twice, once using the implemented outlier detection, and once not using the implemented outlier detection.

The experiments that will be performed are designed to test if and how a schema matching pipeline can be optimized for a given dataset. To test if a hierarchical classification pipeline can improve the matching result we will first test the performance of each individual matching component. After that, we will test if we can optimize the classification result within these single components already by tweaking the configuration for these components. We will then show how a hierarchical classification pipeline can be created and fitted to a dataset. Finally, we will test if this pipeline improves the matching result by comparing it to the initial experiment.

### *A. Dataset*

The dataset that will be used is the CKAN-CERIF dataset. CKAN is an open-source data management system (DMS) for powering data hubs and data portals. This data catalogue system is often used by public institutions to share their data with the general public. The Common European Research Information Format (CERIF) is a data model that allows for a meta-data representation of research entities. Both models are not in csv format. The CKAN dataset consists of XML files while the CERIF dataset consists of RDF files. CKAN model data can be partially mapped to CERIF data. The goal of the experiments performed with these datasets are to research if REPSASM can also be applied to non-column type data structures. It is useful to know this because data integration also often occurs for these (non-column) types of data. The mapping of the CKAN data to the CERIF model has already been done manually.

The XML files are converted using the Parker convention. The Parker convention ignores XML-attributes and simply recreates the XML structure but in json. This conversion was chosen because attributes were not present in the CKAN-XML data. The RDF-files are converted by unpacking the RDF and recursively looping through the RDF-tree, starting at the root, and adding the instances to the json dictionary.

All values in the json dictionaries were removed from the tree-like structures and placed in a column structure by using the path to the tree-leafs as a new column name. This conversion is presented in the following example:

```
json_data = {
main_key: {
key_1 : value_1 ,
key_2 : value_2 }
second_key: {
key_1 : value_3 ,
key_2 : value_4
} }

column_structure = [
(main_key_key_1 , value_1),
(main_key_key_2 , value_2),
(second_key_key_1 , value_3),
(second_key_key_1 , value_4)
]
```

By doing this for all the XML and RDF files and accumulating values with the same column name, we do end up with a column structure. This can be used by REPSASM. Eliminating the tree structure does remove the meaning of each key in the tree. With this experimental setup we'd like to see if the tree-type data is still classifiable based on the tree-leafs, and with that testing if REPSASM can be applied to non-column database data.

The CERIF data is the target schema, and will therefore be used to train a pipeline. The problem however is that a lot of values in the CERIF model are generated from the CKAN data. The generation of data happens during the mapping process and is done by combining multiple elements from the CKAN data into a single CERIF element. The generation of such values is something that REPSASM can not do and is out of scope for this project. To still create a mapping that could be used during the experiments, all column data from both datasets was compared. If two columns from both datasets contained largely the same information, we consider them to be a match, and we give the CKAN column the appropriate label (column name) from the CERIF column. If a column from the CKAN data did not match with any CERIF column we consider it to be an outlier. The mapping and labeling of the CKAN data was done by using all the data, but for the experiments all data is separated before any of the previously discussed conversions (flattening of the tree-structures) were performed into a learn set and a test set. Since the transformation of a tree-structure to a column structure by using the tree-path as a column name often ends up with an abnormally long column name, it was decided that for the experiments and results large column labels will be abstracted using the mapping rules in **Table 1**.

The statistics of the CERIF learn set is shown in **Figure 4**. The statistics of the CKAN test set is shown in **Figure 5**.

## *B. Metrics*

For every experiment we can test two aspects of the framework:

- How well are columns classified (and therefore mapped) when we can assume they are inliers?
- How well are columns classified when we have to deal with outliers?

These two aspects are not only tested to measure the performance of the matchers, but also to measure the influence of outliers on the result. This is important for two reasons. First of all, by testing these two aspects independently we get an indication of how well the outlier detection is working. Secondly, outliers are by far the most occurring class in both datasets, therefore, if the outlier detection does not perform very well, the results will be heavily influenced.

As in many other machine learning applications, the metrics precision, recall, F-measure and accuracy will be used to validate the performance of the pipeline. The metrics are computed using the following outcome variables:

- False Negatives (A): Inlieris detected as an outlier or column is classified incorrectly, and therefore mapped incorrectly.
- True Positives (B): Column is mapped to the correct outcome.
- False Positives (C): Column is classified incorrectly or seen as inlier while it is an outlier.
- True Negatives (D): Column is not part of the dataset and the outlier is correctly detected.

**Table 1:** Column mapping between CKAN and CERIF.

| | |
|---|---|
| A | is_source_of_has_classification_has_term |
| B | is_destination_of_has_source_is_source_of_has_destination_has_URI |
| C | is_dstination_of_has_source_is_source_of_has_destination_type |
| D | is_source_of_has_destination_type |
| E | is_destination_of_has_source_is_source_of_has_endDate |
| F | has_identifier_is_source_of_has_endDate |
| H | has_identifier_has_id_value |
| I | is_destination_of_has_source_has_identifier_has_URI |
| J | is_destination_of_has_source_has_identifier_type |
| K | is_destination_of_type |
| M | is_destination_of_has_source_is_source_of_has_destination_has_name |
| N | is_destination_of_has_source_type |
| O | is_destination_of_has_classification_type |
| P | has_identifier_has_URI |
| Q | is_source_of_has_classification_type |
| R | has_identifier_is_source_of_has_classification_type |
| S | is_destination_of_has_endDate |
| T | is_destination_of_has_startDate |
| V | is_destination_of_has_source_has_identifier_has_id_value |
| X | is_source_of_has_endDate |
| Y | has_identifier_is_source_of_has_startDate |
| a | is_destination_of_has_source_is_source_of_has_classification_type |
| b | is_destination_of_has_source_is_source_of_type |
| c | is_destination_of_has_source_is_source_of_has_startDate |
| d | has_identifier_is_source_of_type |
| e | is_source_of_has_startDate |
| has_description | has_description |
| has_identifier_label | has_identifier_label |
| has_identifier_type | has_identifier_type |
| has_name | has_name |
| is_source_of_type | is_source_of_type |
| label | label |
| type | type |
| unknown | unknown |

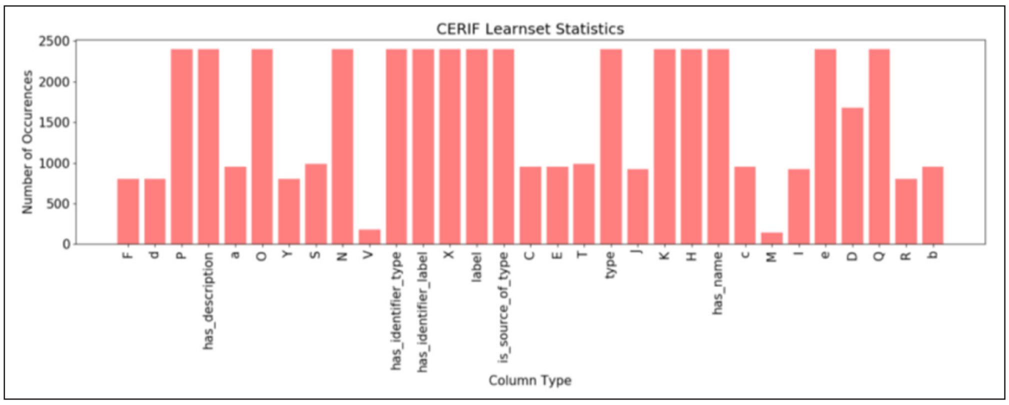

**Figure 4:** Number of instances per class in the CERIF learn set.

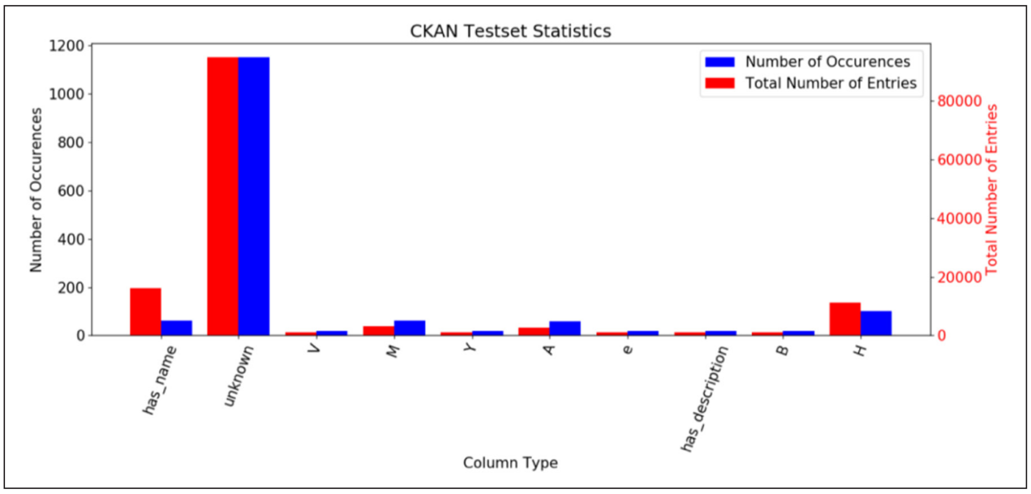

**Figure 5:** Number of instances per class in the CKAN Test set.

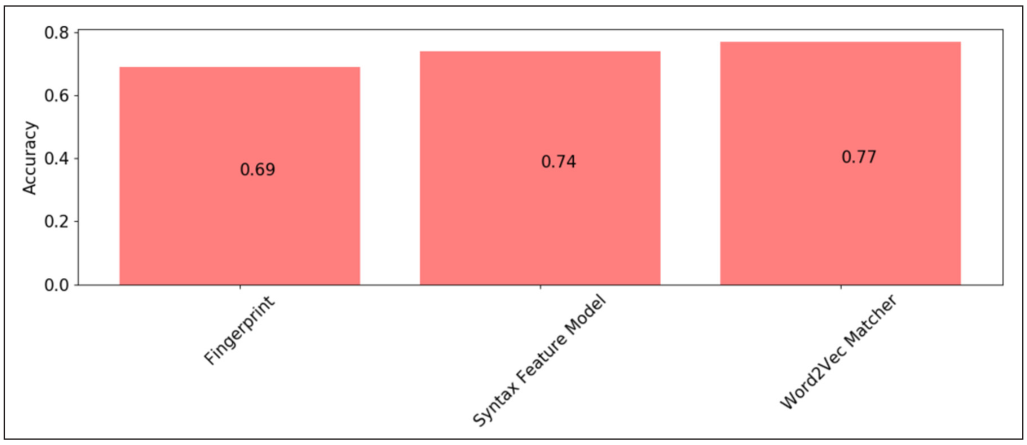

**Figure 6:** Inlier test on the CKAN-CERIF dataset. Accuracy was averaged over 5 tests with 31 classes. Number of simulated columns per class: 15.

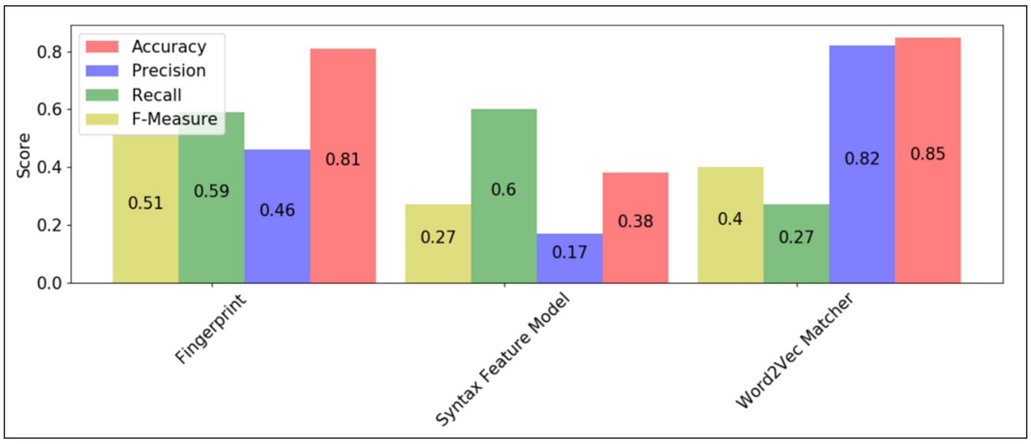

**Figure 7:** Outlier Test on the CKAN-CERIF dataset. Scores were averaged over 5 tests with 31 classes. Number of simulated columns per class: 15.

## *C. Experiment 1: Baseline*

Before experiments with the entire classification pipeline, we will measure the performance of each individual matcher, so we can later determine if building and optimizing a hierarchical classification pipeline improves the matching result.

The result of this experiment will be used as a baseline for further experiments with both datasets. The results of the inlier and outlier experiments are shown in **Figures 6** and **7** respectively. Each test was ran 5 times and the average results are presented.

Here the fingerprint matcher performs the worst, and the Word2vec Model outperforms all matchers. This result could be caused by multiple factors. First of all there is the imbalance in the test-sets to consider. If matchers perform well on the most occurring classes, accuracy will of course be higher. A good example here is the CKAN-CERIF datasets. The *has_name* class, and class H are by far the most occurring in the test set, and the Syntax Feature Model matcher performs very well on distinguishing these exact two classes from the others. The fingerprint matcher performs well on the class V but since this class is less occurring, the confusion in the more prominent classes significantly affects the result.

Accuracy increases when outliers have to be detected for the CKAN-CERIF dataset for the fingerprint matcher. This is because of the imbalance in the test set. Since outliers are often identified correctly, and outliers are the most occurring class in the test set, accuracy will rise because the portion of correctly detection outliers simply outweighs the portion of incorrectly classified inliers. The precision heavily dropped in the CKAN-CERIF experiment when the outlier detection was turned on, and this is the result of the high confusion between outliers and inliers. The both baseline experiments do however show that certain matchers are more capable of classifying specific classes, supporting the hypothesis that piping data to specific matchers could reduce confusion within a classification pipeline. Since not all data is classified correctly, the experiments show that an improvement can be made to the classification process.

## D. Experiment 2: Pipeline

The aim of this experiment is to test the hypothesis that hierarchical classification could improve the matching result. Previous experiments show that outliers heavily influence the results of the experiments due to both their sheer portion of occurrences in the test sets, and the inaccuracy of the implemented outlier detectors. The outliers are only detected at the leaf matchers of the entire match-tree, and until those leaves are reached, data is treated as an inlier, a better outlier detection algorithm could therefore immediately heavily influence the results. Because of this, we will fit the pipeline according to the inlier results, and after this is done measure the overall performance with the outlier detection included.

**Figure 8** shows a representation of the complete and finalized pipeline that was the result of the entire experiment.

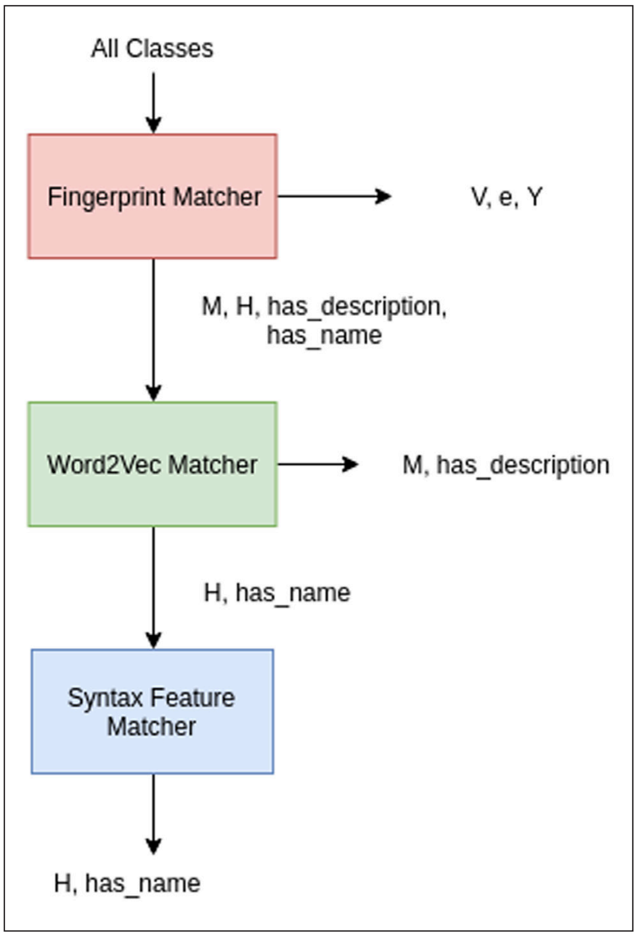

**Figure 8:** Finalized Fitted Pipeline for the CKAN-CERIF dataset.

The following properties can be observed from the CKAN-CERIF data set:

- No distinguishment can made between the classes *e* and *Y*. After looking at their actual class names we could see that both classes consisted of dates, with most of them being the same, therefore we accept in this implementation that this class is classified incorrectly.
- The classes V, e and Y are distinguished perfectly from all others by the Fingerprint matcher.
- The Fingerprint matcher does not classify any of the data as a class that does not occur in the test set, all others do.
- The class has_description is classified perfectly the Word2Vec matcher only holds confusion with classes V and label.
- For class H the SyntaxFeatureMatcher only holds confusion with only class M for the has_name class.

Using these properties we can easily find the optimal matcher configuration. The optimal configuration is as follows:

- First we use the Fingerprint matcher to first extract classes V, e and Y from all others. Only the classes *M, H, has_description*, and *has_name* name are passed on to the next sub-matcher. The Fingerprint matcher has to be used first because it is the only matcher that does not classify any of the data as a class that does not occur in the test set.
- The Word2Vec matcher is used to extract the classes M and has_description. All other data is passed to the next sub-matcher.
- Lastly the Syntax Matcher makes the final classification.

The results of the inlier and outlier experiments are shown in **Figures 9** and **10** respectively. The result depicted in **Figure 10**, is interesting. All scores dropped heavily except for the recall. We can see that outliers are again detected less accurately. The rise in recall is because inliers in the test set are classified more accurately.

Here we also see the downside of applying REPSASM to a flattened tree structure data. The instances of both tree leaves are highly similar but have a different meaning depending on the path to the leaf. REPSASM can not differ between these because it only used the actual leaf data. Perhaps a different feature builder or matcher could have classified it correctly.

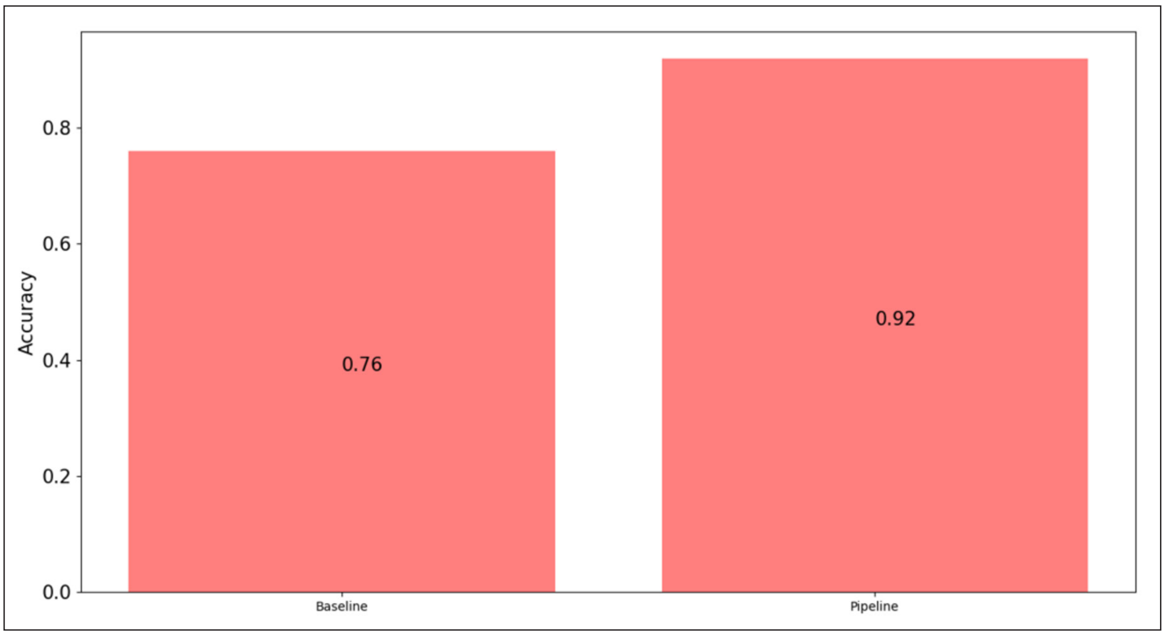

**Figure 9:** Inlier test on the CKAN-CERIF dataset. Accuracy was averaged over 3 tests.

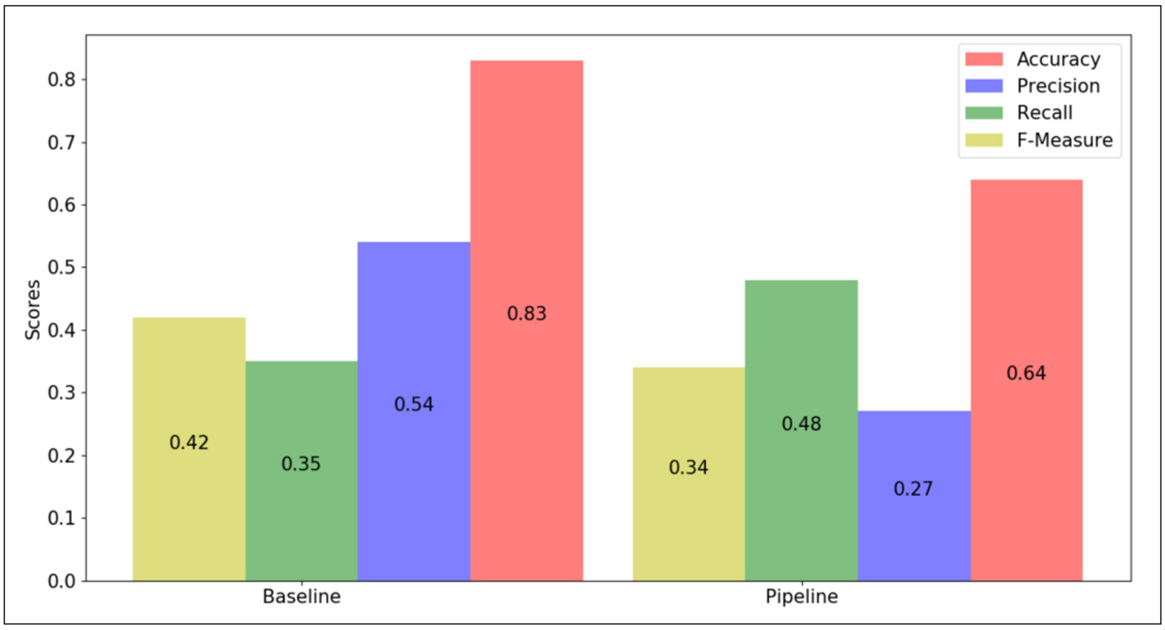

**Figure 10:** Outlier test on the CKAN-CERIF dataset. Scores were averaged over 3 tests.

## V. Conclusion

REPSASM prototype has been designed to allow implementation for all of them so a user can experiment with building a customized pipeline configuration. In order to match schema, a user can first of all create individual matchers based on a single or multiple match criterion (a hybrid matcher). A user can also combine the results of multiple hybrid matchers into a composite matcher. The match criterion can consist of element- or structure-level schema data. The decision making components in these algorithms can base their decision on either a set of rules and constraints, linguistic data, or machine learning. The performance of any pipeline should be measured by the portion of correctly mapped columns or instances. Depending on what the specific goal is, the user could configure the pipeline to have either a higher precision (only predict cases of which you are very certain and classify others as unknown) or a higher recall (predict as much as possible, don't detect outliers). A user should choose his or her optima solution.

The results indicated that collecting the instances in the tree leaves into a column and collectively classifying all the tree leaves together produces good accuracy (91%).

A limitation of the framework is however shown by the CKAN-CERIF dataset: The framework can not handle data that is largely generated during a mapping process.

A limitation which was present in the experiments, but should be investigated in future work, is the detection of outliers. By allowing users to implement their own outlier detection algorithms in the match-tree leafs they can experiment with correctly detecting outliers after their classification pipeline is optimized for all inliers. It is shown that REPSASM can aid in creating an optimized schema-matching pipeline by enabling a user to implement sub-matchers for classes that contain confusion among each other.

The framework can also finds its use outside of a schema matching context. Users could use the configurable pipeline structure to experiment with optimizing hierarchical classification within any environment. The contribution of REPSASM is also more valuable if the pipeline structure can be used for experimentation outside of the schema matching context. Even though actually utilizing REPSASM for this purpose is out of scope for this research project, it should still be possible because it adds more value to the contribution. Therefore, creating and configuring the pipeline should be independent of utilizing it for schema matching.

## Acknowledgement

This research is supported by the European Unions Horizon 2020 research and innovation program under grant agreements 654182 (EN-VRIPLUS project), 824068 (ENVRI-FAIR), and 825134 (ARTICONF). The first author also acknowledge the support from National Natural Science Foundation of China under grant agreement 61762062.

## Competing Interests

The authors have no competing interests to declare.

## References

**Algergawy, A, Nayak, R** and **Saake, G.** 2010. Element similarity measures in xml schema matching. *Information Sciences* 180(24): 4975–4998. DOI: https://doi.org/10.1016/j.ins.2010.08.022

**Amann, B, Fundulaki, I, Scholl, M, Beeri, C** and **Vercoustre, A-M.** 2001. Mapping xml fragments to community web ontologies. *WebDB 2001-International Workshop on the Web and Databases*, pp. 97–102.

**Berlin, J** and **Motro, A.** 2002. Database schema matching using machine learning with feature selection. *International Conference on Advanced Information Systems Engineering,* Springer, pp. 452–466.

**Bex, GJ, Neven, F** and **Van den Bussche, J.** 2004. Dtds versus xml schema: a practical study. *Proceedings of the 7th international workshop on the web and databases: colocated with ACM SIGMOD/PODS 2004,* ACM, pp. 79–84. DOI: https://doi.org/10.1145/1017074.1017095

**Biron, PV, Malhotra, A, WWW Consortium,** et al. 2004. Xml schema part 2: Datatypes.

**Bischof, S, Decker, S, Krennwallner, T, Lopes, N** and **Polleres, A.** 2012. Mapping between rdf and xml with xsparql. *Journal on Data Semantics* 1(3): 147–185. DOI: https://doi.org/10.1007/s13740-012-0008-7

**Breitling, F.** 2009. A standard transformation from xml to rdf via xslt. *Astronomische Nachrichten: Astronomical Notes* 330(7): 755–760. DOI: https://doi.org/10.1002/asna.200811233

**Brickley, D, Guha, RV** and **McBride, B.** 2014. Rdf schema 1.1. *W3C recommendation* 25: 2004–2014.

**Davy Van, D, Chris, P, Gaëtan, M, Erik, M** and **Rik Van de, W.** 2008. Xml to rdf conversion: A generic approach. *2008 International Conference on Automated Solutions for Cross Media Content and Multi-Channel Distribution*, pp. 138–144.

**Do, H-H** and **Rahm, E.** 2002. Coma: A system for flexible combination of schema matching approaches. *Proceedings of the 28th International Conference on Very Large Data Bases, VLDB '02*, VLDB Endowment, pp. 610–621. URL: http://dl.acm.org/citation.cfm?id=1287369.1287422. DOI: https://doi.org/10.1016/B978-155860869-6/50060-3

**Garcia-Gonzalez, H** and **Labra-Gayo, JE.** 2018. Xmlschema2shex: Converting xml validation to rdf validation. *Semantic Web* (Preprint): 1–19. DOI: https://doi.org/10.3233/SW-180329

**Klein, M.** 2002. Interpreting xml documents via an rdf schema ontology, Database and expert systems applications. *2002 proceedings 13th international workshop on IEEE*, pp. 889–893.

**Li, W-S** and **Clifton, C.** 2000. Semint: A tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data & Knowledge Engineering* 33(1): 49–84. DOI: https://doi.org/10.1016/S0169-023X(99)00044-0

**McGuinness, DL, Van Harmelen, F,** et al. 2004. Owl web ontology language overview. *W3C recommendation* 10(10): 2004.

**Mikolov, T, Sutskever, I, Chen, K, Corrado, GS** and **Dean, J.** 2013. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems,* pp. 3111–3119.

**Miller, GA.** 1995. Wordnet: a lexical database for english. *Communications of the ACM* 38(11): 39–41. DOI: https://doi.org/10.1145/219717.219748

**Nadeau, D** and **Sekine, S.** 2007. A survey of named entity recognition and classification. *Lingvisticae Investigationes* 30(1): 3–26. DOI: https://doi.org/10.1075/li.30.1.03nad

**Nayak, R** and **Tran, T.** 2007. A progressive clustering algorithm to group the xml data by structural and semantic similarity. *International Journal of Pattern Recognition and Artificial Intelligence* 21(4): 723–743. DOI: https://doi.org/10.1142/S0218001407005648

**Prud'hommeaux, E, Labra Gayo, JE** and **Solbrig, H.** 2014. Shape expressions: an rdf validation and transformation language. *Proceedings of the 10th International Conference on Semantic Systems,* ACM, pp. 32–40. DOI: https://doi.org/10.1145/2660517.2660523

**Rahm, E** and **Bernstein, PA.** 2001. A survey of approaches to automatic schema matching. *The VLDB Journal* 10(4): 334–350. DOI: https://doi.org/10.1007/s007780100057

**Silla, CN** and **Freitas, AA.** 2011. A survey of hierarchical classification across different application domains. *Data Mining and Knowledge Discovery* 22(1): 31–72. DOI: https://doi.org/10.1007/s10618-010-0175-9

**Sun, A** and **Lim, E-P.** 2001. Hierarchical text classification and evaluation. *Proceedings 2001 IEEE International Conference on Data Mining*, pp. 521–528.

**Tao, J, Sirin, E, Bao, J** and **McGuinness, DL.** 2010. Integrity constraints in owl. *AAAI.*

**Thuy, PTT, Lee, Y-K** and **Lee, S.** 2012. S-trans: Semantic transformation of xml healthcare data into owl ontology. *Knowledge-Based Systems* 35: 349–356. DOI: https://doi.org/10.1016/j.knosys.2012.04.009

**Thuy, PTT, Lee, Y-K** and **Lee, S.** 2013. A semantic approach for transforming xml data into rdf ontology. *Wireless Personal Communications* 73(4): 1387–1402. DOI: https://doi.org/10.1007/s11277-013-1256-z

**Thuy, PTT, Lee, Y-K, Lee, S** and **Jeong, B-S.** 2007. Transforming valid xml documents into rdf via rdf schema. *Next Generation Web Services Practices, 2007. NWeSP 2007. Third International Conference on,* IEEE, pp. 35–40. DOI: https://doi.org/10.1109/NWESP.2007.23

**Yang, D** and **Powers, DM.** 2005. Measuring semantic similarity in the taxonomy of wordnet. *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*, Australian Computer Society, Inc., pp. 315–322.

**How to cite this article:** Liao, X, Bottelier, J and Zhao, Z. 2019. A Column Styled Composable Schema Matcher for Semantic Data-Types. *Data Science Journal,* 18: 25, pp. 1–15. DOI: https://doi.org/10.5334/dsj-2019-025

**Submitted:** 23 February 2019 **Accepted:** 14 May 2019 **Published:** 24 June 2019