TAPON-MT: a versatile framework for semantic labelling

Daniel Ayala[a,*], Inma Hernández[a], David Ruiz[a], Miguel Toro[a]

[a] *Universidad de Sevilla, ETSI Informática.*
Avda. de la Reina Mercedes, s/n, Sevilla E-41012, Spain.

Abstract

Semantic labelling refers to the problem of assigning known labels to the elements of structured information from a source such as an HTML table or an RDF dump with unknown semantics. In the recent years it has become progressively more relevant due to the growth of available structured information in the Web of data that need to be labelled in order to integrate it in data systems. The existing approaches for semantic labelling have several drawbacks that make them unappealing if not impossible to use in certain scenarios: not accepting nested structures as input, being unable to label structural elements, not being customisable, requiring groups of instances when labelling, requiring matching instances to named entities in a knowledge base, not detecting numeric data, or not supporting complex features. In this article, we propose TAPON-MT, a framework for machine learning semantic labelling. Our framework does not have the former limitations, which makes it domain-independent and customisable. We have implemented it with a graphical interface that eases the creation and analysis of models, and we offer a web service API for their application. We have also validated it with a subset of the National Science Foundation awards dataset, and our conclusion is that TAPON-MT creates models to label information that are effective and efficient in practice.

Keywords: Semantic labelling, Information Integration, Machine Learning

1. Introduction

Semantic labelling refers to the problem of analysing structured information and endowing it with labels that denote known classes from an ontology[22]. A dataset of structured information, usually extracted from the web in a crawling process [8] is taken as input, and each element in the dataset is assigned one or several labels, which correspond to the classes that best describe the element according to its features. Semantic labelling is a cornerstone for the integration of information from heterogeneous data sources, which is common in several contexts[3, 12, 19], and can be used in tasks related to information extraction[2, 20] (by using the labels to determine whether or not a piece of information should be extracted), information verification[11, 13, 15] (by using the labels to confirm the assumed class of an element), or ontology matching[6] (by using the labels to study the similarities between pairs of classes). The format of the input information can range from simple HTML tables to complex structures in RDF datasets. Regardless of their source format, we can represent information with generic structures as exemplified in Figure 1, which depicts information about an award from the NSF awards dataset[7]: namely an award record, which is a structural element with no associated text, containing other records and a number of attributes, which are textual elements (elements with an associated string, which includes numbers, dates, etc.). Both records and attributes are instances of a number of classes such as "title" or "awardee", though the class of instances can be unknown, which motivates their labelling.

Semantic labelling can be seen as a supervised classification process in which the input is one record or attribute, and the output is one or more labels that correspond to the semantic classes that best describe them. Note that the need for supervision is not a problem, thanks to the large amount of available labelled, structured information in the

*Corresponding author
Email addresses: dayala1@us.es (Daniel Ayala), inmahernandez@us.es (Inma Hernández), druiz@us.es (David Ruiz), migueltoro@us.es (Miguel Toro)

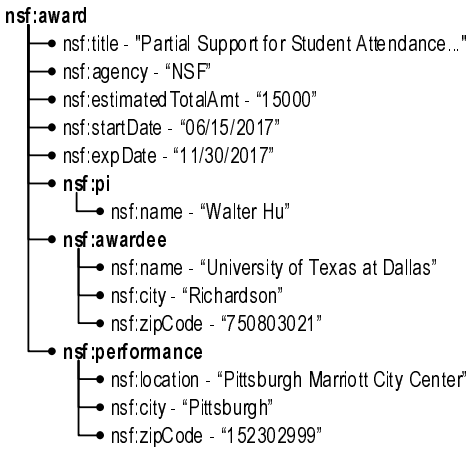


Figure 1: Structured information example

Web of Data[4] that can be used to train the models. The selection of labels is based on features that measure the similarity between the input and the known classes, such as the number of digits in a textual value[11, 13], or the score of a knowledge base query results[17, 23, 25, 22]. What specific features are used by each approach greatly affects its accuracy. For example, one that only measures the number of uppercase and lowercase letters will not be able to properly classify attributes that are not characterised by those features, such as prices, telephone numbers, or emails.

The area of automated classification has been extensively researched in the past[9], and many techniques have been proposed that deal with this problem. Obviously, when used as a part of a semantic labelling process, the performance of these techniques has a significant effect on the accuracy of the labelling. However, we consider the study of these techniques to be beyond the scope of the paper, since they are completely independent of semantic labelling approaches that apply them. Instead, we focus on aspects that are specific to semantic labelling.

Regarding proposals in the related work, they have been devised with specific application scenarios in mind, and thus lack in versatility. They can usually be only applied suitably under certain conditions, and while focusing on certain scenarios helps them achieve better results under their conditions, they lack in applicability under different ones. The following limitations are common among them: they can not be applied to structures with nested records, only to flat ones; they can only be used to label attributes; they rely on a particular classification technique and a specific set of features; they can not label individual instances, but only groups; they rely on matching instances to named entities in a knowledge base; they do not detect numeric data; and they do not support complex features that are necessary to model some classes.

Figure 1 exemplifies these problematic cases. There are records inside other records, so approaches that only label flat records would obtain poor labels, if any, without computationally expensive flattening preprocessing that may not be viable. There are both records and attributes, so approaches that only label attributes would not provide a complete labelling. Several approaches that label groups of instances use the distribution of a group of numeric attributes to infer their class, but they would be unable to label the "estimatedTotalAmt" attribute, since there is a single instance being classified. Several attributes, such as "estimatedTotalAmt", or "startDate" do not correspond to named entities and can not be found in a knowledge base. There are several attributes with numeric data: "estimatedTotalAmt" and "zipCode", which can only be properly processed if the approach detects that they are numeric and uses their value during the labelling process. Finally, there are some pairs of instances that are nearly indistinguishable and require complex features: the "startDate" and "expDate" attributes, which have an identical format, and the "awardee" and "performance" records, which have an identical structure.

In order to solve these problems, we have created TAPON-MT, a new semantic labelling framework whose architecture has been devised with applicability and customisation in mind, and is based on a two-phase classification technique of our own[1]. Applicability implies that our framework is domain-independent and does not rely on specific conditions. Customisation implies that the architecture and workflow of our framework give the possibility to integrate a wide variety of features and classification techniques. The main strengths of our framework are that it supports nested records as input; it supports the labelling of both records and attributes; its features and classification techniques can be customised; it labels structures individually; it does not rely on entity matching; it detects numeric data; and its architecture supports several kinds of complex features, including dynamically generated features, and hint-based features (a kind of feature that requires a two-phase process). Re-

garding the example in Figure 1, TAPON-MT is able to label all the records and attributes in the nested structure, to label the "estimatedTotalAmt" attribute without other instances of the same class, to label the attributes that do not correspond to named entities, to detect what attributes are numeric and leverage the use of features based on their values, and to label the nearly indistinguishable instances thanks to one of the supported complex features: hint-based features, which allow TAPON-MT to learn that "startDate" attributes are near something that resembles a date but have a lower value than it, while "expDate" attributes have a higher value than it. Similarly, they would allow it to learn that "awardee" records contain an attribute that seems to be a name, while "performance" records contain an attribute that seems to be a location.

Our framework provides two different interfaces that make it usable both as a web service API and as a desktop application with a graphical interface. The web service takes a dataset in JSON format, and the response contains the labels. The graphical interface allows the user not only to create models, but also to visualise and validate them. It can be used to create models by choosing features, classification settings and data sources; used to visualise the created models and check for suspicious classifiers; and used to test the models by selecting the data it is applied to, obtaining several measurements of performance, as well as a matrix that can be used to study the similarity between classes. In addition to these implementations, we have performed ten-fold cross validation to confirm that models created with our framework are able to achieve good results by comparing it to the results obtained by implementations of the proposals by Kushmerick [11], Ramnandan et al. [22], and Pham et al. [21] The experiments consisted in labelling information from the National Science Foundation RDF dataset.

The rest of the article is organised as follows: Section 2 describes the analysis of the relevant approaches we have identified in the literature; Section 3 reports on some preliminaries that are necessary to understand our framework; Section 4 contains a detailed description of TAPON-MT, including the underlying data model and architecture; Section 5 describes how we have validated our framework by implementing it and performing experiments; finally, Section 6 recaps on our main conclusions.

2. Related work

In this section, we present existing proposals that are related to TAPON-MT. We present both approaches for semantic labelling and approaches for wrapper verification, which check the information extracted by a web wrapper to ensure that it is still working properly. The latter share the same principles as semantic labelling ones, since they need to create a model for valid information classes and confirm that the extracted information is classified as valid according to that model. This makes them relevant to semantic labelling and our study of the literature. In several cases, they can easily be used for semantic labelling with slight modifications.

2.1. Semantic labelling

Semantic labelling proposals aim to label structured information with labels that denote its semantics in reference to a known schema, which can range from the columns of a local table to the elements of an OWL ontology. Many of these techniques focus on semantic labelling of tables, where rows can be seen as records, and columns as attributes.

Limaye et al. [14]'s technique learns a model from a catalog: a hierarchy of types, entities that instance them, and binary relationships between types. The model is used to label web tables by means of a maximisation algorithm, including the labelling of columns, cells, and relationships between columns. Limaye et al. define an objective function, which measures how consistent a set of labels is according to the catalog. For example, a set of labels is more consistent if the text of a column header is similar to the label of the column. The variables of the objective function are the labels. Approximation algorithms are used to reduce the large search space.

Venetis et al. [24]'s technique labels columns and relationships between columns in a table, outputting several labels for each element. It requires two knowledge bases: a collection of entities (represented by a piece of text) and the class to which they belong, and a collection of relationships between the entities. These are obtained through web crawling, using textual patterns. In order to label columns, Venetis et al. use a bayesian classifier which computes the probability of each label being correct, by checking that the cell values have the same type in the knowledge base (there must be an exact match of the entities and the cell values).

Mulwad et al. [17]'s technique labels cells, columns, and relationships between columns in a table by means of a message propagation algorithm. Columns and cells are represented by nodes in a graph. The nodes are then connected to additional nodes called factor nodes, which represent a group of related labels that should be consistent, such as the label of a column and the labels of the cells it contains. For example, if a column is labelled as "cities", the cells it contains should be labelled as entities of class "city". Starting with a set of candidate labels for each node, obtained from queries to Wikitology, each factor node computes the consistency of the labels that are passed to it and, if necessary, requests nodes to change their label. This process is repeated until there is convergence and no further changes are requested.

Ritze et al. [23]'s tool, T2K Match, labels rows and columns in HTML tables. First, it selects, for each row in the table, a group of candidate entities obtained by searching in DBpedia for the entity label, that is, the content of the subject column (the column that denotes the overall entity represented by a row), which is considered to be the column with the highest number of unique values (taking the leftmost column in the case of a tie). The most frequent classes among the retrieved entities are selected. Candidate entities that do not belong to these classes are discarded and new searches are performed with the selected classes as an additional constraint. Rows are then labelled with the most similar candidate entity. Then, columns are given a label by comparing the similarity of their values to the properties of row candidates. Finally, an iterative process is used to refine the labels of rows and columns.

Zhang [25]'s tool, TableMiner+, labels columns, cells and relationships between columns in HTML tables. First, the subject column is detected. Then, columns that contain named entities are given preliminary labels in an iterative process based on candidate entities retrieved from a knowledge base using queries, which do not only include the text of cells in the column, but also the text from nearby cells and text outside the table in the HTML document. In the next step, their cells are labelled with entities using the column label as a constraint. Another iterative process is used to update column and cell labels and ensure they are consistent. Finally, relationships between columns are detected, and columns that do not contain named entities are labelled.

Ramnandan et al. [22]'s technique labels bags of attributes (that is, groups of attributes that are known to share the same class, without the structure in which they are contained being considered) by using a set of already labelled examples (which consists of value-class pairs) stored in a Lucene index. Using textual patterns, if a class in the labelled examples is considered to be textual, all the examples are concatenated and stored as a single document in the index, associated to the class. If the class is numeric, the examples are stored individually. When an unlabelled set of attributes has to be labelled, if they are textual, they are concatenated and used to query the index, so that the document with the highest score (based on TF-IDF similarity) among the results represents the chosen label. If they are numeric, distribution equivalence tests are used to determine the most similar class.

Pham et al. [21]'s technique builds on that of Ramnandan et al. The TF-IDF and distribution similarity are expanded with other similarity metrics: the textual and numeric Jaccard similarity, the attribute class name similarity, and the histogram similarity (for numeric attributes). These metrics are used to create feature vectors associated to a pair of sets of attributes. Each features vector can be classified as "True" or "False", the former meaning that the two sets of attributes belong to the same class. The final output is the top-k attribute classes whose corresponding vectors were classified as "True", sorted by the probability of the classification, used as a confidence score.

Neumaier et al. [18]'s approach, like the former, labels bags of attributes, but focuses exclusively on numeric attributes and multi-level semantic labelling (endowing information with several classes, corresponding to progressively more detailed levels in the ontology). It is trained by building a graph. This graph contains several trees, each of them corresponding to an attribute class. The root of each of these trees encompasses all attribute examples of the class (e.g. "height"), and nodes in deeper layers further refine the context of the attribute (e.g. "height of a person", and "height of a player"), having less attributes associated to them. Nearest neighbour classification is used to label a new set of unlabelled attributes, measuring their distance to the nodes of the graph using two distance measures: the inverse distribution similarity and the euclidean distance between vectors containing descriptive measures such as the maximum and minimum of the values.

2.2. Wrapper verification

The following proposals were not devised to be able to endow structured information with semantics. However, they model this information in a similar way by computing features related to format and structure. They aim to check that a web wrapper that extracts information from a website has not stopped working properly because of undetected changes in said website. As an example, a wrapper that is based on the rule "extract the content of the third tag as the article's price" is vulnerable to changes in the HTML code of the website pages, and a change in style that rearranges the HTML elements could lead the wrapper to start extracting wrong information, such as the text "BUY" as the articles' price. In order to verify whether or not this has happened, they could check that the piece of text that has been extracted as a price truly seems to be a price according to a model. The difference with semantic labelling is that the input already has a label, which is confirmed.

Kushmerick [10, 11]'s tool, RAPTURE, compares newly extracted datasets form a web wrapper that is applied to a website to a collection of pre-verified datasets from the same website. RAPTURE checks that the value of the features of the instances of a class are not anomalous considering the distribution of feature values observed in the pre-verified dataset, such as the number of characters in product prices. In order to do this, the value of each feature for each class is modelled using a normal distribution. When a new dataset has to be verified, the feature values are computed for each instance of each of its classes. If, with regards to a certain class, the combined probabilities of its instances are below a given threshold, the dataset is considered to be anomalous and potentially erroneous. Although this verification only involves attributes, RAPTURE also includes basic record verification based on probabilities propagation.

Lerman et al. [13]'s tool, DataProG, is based on features such as the number of occurrences of certain tokens types in the attribute values. These features also include patterns that describe common beginnings and endings of attribute values. From a pre-verified datasets, the average value of these features is computed and stored in a vector for each attribute class. When a new dataset has to be verified, a feature vector is computed for each class and compared to the pre-verified one. If the vectors are considered to be too distant using the χ^2 goodness-of-fit test, the dataset is considered anomalous.

McCann et al. [15]'s tool, Maveric, models a set of features as normal distributions in a similar way to Kushmerick, but introduces several improvements: probabilities are normalised to avoid disproportionally low probabilities in flat distributions, and perturbations are used to introduce examples of invalid data. These perturbations are crafted manually, and are used to infer a second set of normal distributions that represent invalid data. High probabilities in these contribute towards the dataset being considered anomalous.

Approaches	F_1	F_2	F_3	F_4	F_5	F_6	F_7
Wrapper verification							
Kushmerick [10]	✓	∼	✓	✓	✓	×	×
Lerman et al. [13]	×	×	✓	✓	✓	×	✓
McCann et al. [15]	×	×	✓	✓	✓	∼	×
Semantic labelling							
Limaye et al. [14]	×	✓	×	×	×	×	×
Venetis et al. [24]	×	✓	×	×	×	×	×
Mulwad et al. [17]	×	✓	×	×	×	×	×
Ritze et al. [23]	×	✓	×	×	×	✓	×
Zhang [25]	×	✓	×	×	×	×	✓
Ramnandan et al. [22]	✓	×	×	×	✓	✓	×
Pham et al. [21]	✓	×	✓	×	✓	✓	✓
Neumaier et al. [18]	✓	×	✓	×	✓	×	×
TAPON-MT	✓	✓	✓	✓	✓	✓	✓

F_1 = Nested records; F_2 = Labels records and attributes; F_3 = Customizable features and techniques; F_4 = Individual instance labelling; F_5 = Does not require entity matching; F_6 = Numeric data detection; F_7 = Complex features;

Table 1: Comparison of related work.

2.3. Discussion

Table 1 summarises the comparison of the related work according to a number of features related to applicability and customisation. In this table, the ✓ symbol denotes that the approach supports a feature, symbol × denotes that the approach does not support a feature, and ∼ symbol entails that the feature is partially supported by the approach. The features we have analysed are the following:

F_1**:** This feature determines if it is able to process structures with nested records, as opposed to only being able to process flat ones.

F_2: This feature determines if it models and labels both records and attributes, not being limited to only attributes or records.

F_3: This feature determines if it can be customised with different features or techniques.

F_4: This feature determines if it assigns each label to an individual instance as opposed to a group of instances.

F_5: This feature determines if it does not require matching instances to named entities in a knowledge base.

F_6: This feature determines if it is able to distinguish numeric attributes from textual ones and treat them separately.

F_7: This feature determines if it supports the application of complex features. That is, it supports features that require a specific architecture.

Regarding F_1 (nested records), approaches that focus exclusively on flat structures are much more limited than those that can deal with more complex structures. This is specially relevant given the increase in popularity of formats such as RDF or JSON[16], which have deep, arbitrary structures. Most approaches are limited to tuples that do not contain nested records with deep structures. Among wrapper verification proposals, Kushmerick [11] is the only one to consider the possibility of nested records that need to be verified, and offers a simple way to propagate probabilities from instances to the records that contain them. Semantic labelling proposals that label tables are limited to flat records in the form of rows. TAPON-MT supports nested records.

Regarding F_2 (labels records and attributes), not being able to label records or attributes greatly hinders the applicability of a solution, which would only be able to partially label datasets. Many approaches focus exclusively on attributes, and are therefore unable to label records. Wrapper verification proposals only model attributes. The semantic labelling proposals by Ramnandan et al. [22], Pham et al. [21] and Neumaier et al. [18] label bags of attributes, and even though they accept as input complex structures that may include both records and attributes, they can not label the records. Neumaier et al. tries to partially circumvent this limitation by means of multi-level labelling, including the record class in the attribute label. For example, instead of labelling an attribute as a zipCode, it would label it as an awardee-zipCode. TAPON-MT labels both attributes and records.

Regarding F_3 (customisable features and techniques), being able to accept new features and classification techniques makes a solution able to integrate new ways to label information and enrich the models when the used features are not enough. For example, the technique by Ramnandan et al. [22] relies on a process that involves querying a lucene index and keeping the result with the highest score. In this process there is no freedom to include additional features or different classification techniques that may be useful in complex scenarios. TAPON-MT applies features and classification techniques in a modular, easy to customise way.

Regarding F_4 (individual instance labelling), approaches that label groups of instances assuming they belong to the same class do not work properly when the input has individual instances that are not known to share the same class. This is the case when datasets are labelled individually, instead of labelling several datasets with aligned instances that follow a known schema. Datasets with unknown schemas, optional fields and fields of variable length are common in web environments[5] The techniques by Ramnandan et al. [22], Pham et al. [21] and Neumaier et al. [18] expect as input a group of several attributes that share the same class. All proposals that focus on tables also make this assumption, since all rows of the same table (which would be records) are known to share the same class. Similarly, all the cells of the same column (which would be attributes) share the same class as well. TAPON-MT labels both records and attributes individually (that is, every instance gets an individual label).

Regarding F_5 (does not require entity matching), approaches that either only label attributes that correspond to named entities (such as the name of a city) or records that correspond to named entities (such as the author of a publication) are unable to label many instances from real-world contexts that do not have entries that represent them in a knowledge base. For example, attributes may correspond to information such as prices, heights, or enumerates. Records may, too, correspond to dates, time intervals, or addresses. Most of the existing semantic labelling proposals are tied in one way or another to entity matching, since their labelling process involves in one way or another mapping instances to

entities in a knowledge base. TAPON-MT does not rely on any kind of named entity matching.

Regarding F_6 (numeric data detection), numeric attributes offer information (their numeric value) that can be taken into account to improve the model with additional features. Being able to detect such attributes and handle them accordingly is crucial when there are several numeric classes whose only difference is the distribution of their values or other similar aspects. Therefore, it is mandatory to analyse the numeric values of the instances of these classes in order to correctly classify them. Even though identifying a piece of text as numeric is trivial, only three proposals integrate this distinction into their workflow. Note that Neumaier et al. [18] only labels numeric attributes, so they do not support this feature, which would require the distinction of numeric attributes from texual ones and treating each group accordingly, supporting both cases. TAPON-MT detects what attributes are numeric, as well as what classes in the training set are (those that contain a fraction of numeric attributes above a user-given threshold). The numerical value of attributes and the information about what classes are numeric are stored so that features or classification techniques can use them in any way.

Regarding F_7 (complex features), supporting complex features that require a specific architecture or process helps model complex cases where typical features are not enough to tell some classes apart, and additional information is needed. Lerman et al. [13] support features based on starting and ending patterns, Zhang [25] supports features from the context of a table, and Pham et al. [21] support similarity features. TAPON-MT supports similarity features, dynamically generated parametrical features, incremental features, and hint-based features.

3. Preliminaries

In this Section, we introduce definitions of key concepts that are necessary to understand our framework, including a short description of important aspects. Said concepts are depicted in Figure 2.

Dataset: a collection of instances of semantic classes (or "classes", for short).

Class: a piece of text that denotes the actual semantics of information, such as "book", "price" or "author".

Label: the predicted class of an instance. Ideally, the label is the same as the actual class. Note that both classes and labels are not related to datatypes such as Integer, String or Boolean.

Hint: a temporary label used in iterative techniques.

Record: a structural instance with no text associated to it. It may contain other instances.

Attribute: an instance with a piece of text associated to it (its textual value). It does not contain other instances.

Feature: a property that can be computed from some element. The domain of a feature is the kind of element it can be computed from. For example, the feature "number of uppercase letters" can only be computed from attributes. The feature "number of contained attributes" can be computed from records. The feature "distance to the nearest price" can be computed from both.

Featurable : an element from which features can be computed. Instances are featurables, since we can compute a variety of features such as the occurrence of textual patterns, the number of attributes of a record, etc. Datasets are featurables too, since we can also compute features associated to the entire dataset, such as the number of instances it contains, or the average value of a feature among its instances (the usefulness of these features will be discussed later). Each featurable may have a numeric value for each of the features.

HB-Feature (hint-based feature): a feature that, in order to be computed, requires the instances in the dataset to have hints.

FH-Feature (hint-free feature): any feature that is not a HB-Feature.

Model : a representation of classes that is trained from a set of featurables used as examples, and is able to output labels for an instance. A model is tied to the features from its training set.

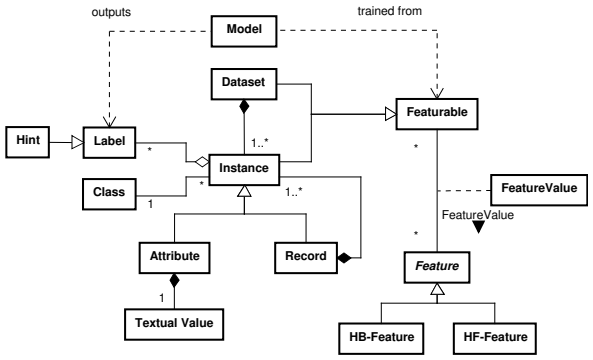


Figure 2: Conceptual model.

4. TAPON-MT

4.1. Workflow

Our framework takes as input a dataset, and it outputs labels for each record and attribute individually, providing them with either one single label or several labels ranked by probability. This can be done through a desktop interface or a deployed web service, as described in Section 5.1. Common web information formats, such as JSON files, HTML tables, CSV files or RDF data can be parsed and transformed into our generic input structures, which is trivial in most cases. For example, to parse a CSV file, each row would be a record, and commas would separate the different values of the attributes.

To illustrate the labelling of datasets using TAPON-MT, we use the dataset in Figure 1 as a running example.

TAPON-MT is based on a two-phase machine learning approach that trains a pair of classification models from a number of training datasets, and then is able to apply them consecutively to label data as described in our previous work [1]. An overview of the training process is depicted in Figure 3. It comprises six steps, mainly: (1) "Features Calculator setup" involves iterating through every instance in the training datasets, storing the textual values of each attribute class in an index, and determining what features will be computed (as we later describe in detail). (2) "Compute HF feature vectors" computes the HF features, and stores the feature vectors of attributes and records in CSV files. (3) "Train HF model" uses these vectors to train the classifiers of a model that only uses HF features. (4) "Label TD with HF model" applies the HF model to the training datasets, obtaining a set of hints (we use these to compute HB features,

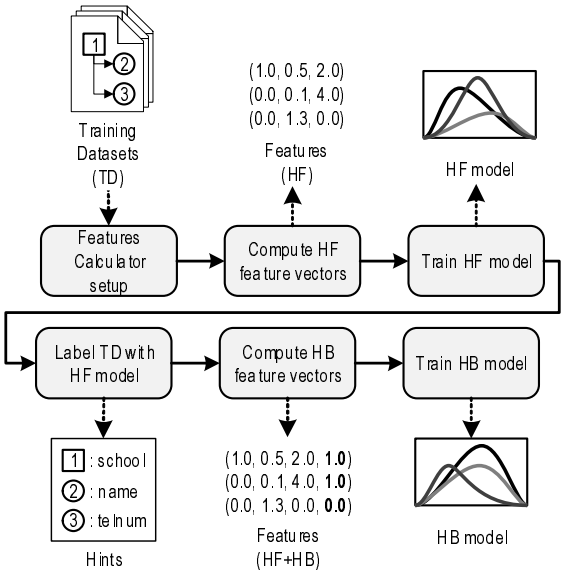


Figure 3: Overview of training TAPON-MT.

not the actual classes, which are only known in the training set). (5) "Compute HB feature vectors" expands the former feature vectors with new features that are based on hints (HB features), and stores them in additional CSV files. (6) "Train HB model" uses said vectors to train a second model that uses both HF and HB features.

TAPON-MT can also be applied without hint-based features, in which case steps 4, 5, and 6 would not be needed.

The application of the trained models to an input dataset is very similar to the former training process: first, HF features are computed, the HF model is used to obtain a set of hints, HB features are computed using the hints, and finally, the HB model is used to obtain the output labels.

4.2. Architecture

Figure 4 presents the architecture of our framework. It comprises two components: first, the main TAPON-MT component, that includes the models handler that trains and applies models, the graphical interface and the Web service provider, as well as a DAO layer used to access external sources of data and deal with persistence. The second component, the features framework, takes care of computing features from datasets, as well as, if needed, storing the resulting vectors in persistent files, which can be seen as tables where each row corresponds to an instance, and each column to

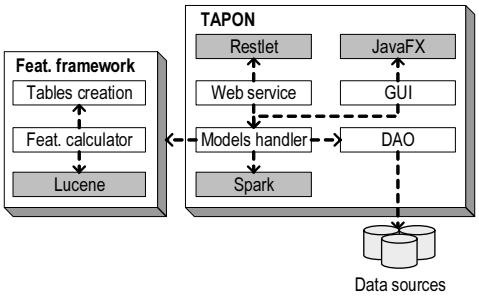


Figure 4: Component diagram of TAPON-MT.

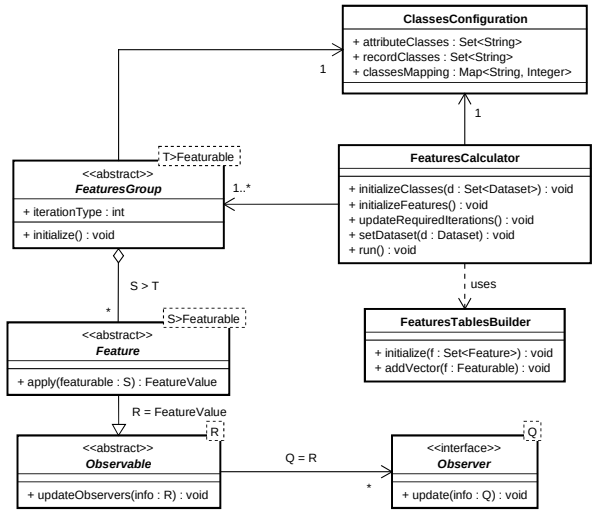


Figure 5: Class diagram of the features framework.

a feature. We used the Spark framework to support the creation of models using a large number of datasets in a scalable way, since models can be trained using arbitrarily large files.

Next, we first describe the features framework in Section 4.2.1; then, we describe the models handler in Section 4.2.2.

4.2.1. Features framework

The architecture of the features framework is presented in Figure 5. The features calculator is able to compute features from records, attributes, and entire datasets. Features can be easily added or removed.

Class "FeaturesCalculator" provides the methods that are used to perform set-up and compute features from a dataset.

Method "initializeClasses" takes a set of training datasets, iterates over them, identifies the set of semantic classes that are present in them, and passes it to class "ClassesConfiguration", which stores what semantic classes are known to exist and maps them to integers that represent them from the point of view of classifiers. Additionally, each example of each attribute semantic class is stored in a Lucene index. These examples are used to compute some features that require them. One example of such a feature could be "the average edit distance to stored examples of semantic class title".

Method "initializeFeatures" creates the objects that represent features, that is, that instantiate classes that extend the abstract class "Feature" and have an "apply" method that computes a feature value associated to a featurable. One example of such class could be "NumberOfCharacters", an attribute feature that computes the number of occurrences of a character type, represented by a pattern such as "\pLu" (unicode uppercase letters) or " \pS" (unicode symbols). The pattern is a parameter that can take a different value in each "NumberOfCharacters" object. The creation of several objects that correspond to the same feature with different parameter values is done by features groups: classes that extend "FeaturesGroup" and dynamically create several instances of features during setup, having access to the "ClassesConfiguration" object in case it is needed to create the objects. For example, features group "AverageSimilarityGroup" would create objects that instantiate feature "AverageSimilarity", that measures to the average similarity to a set of examples of an attribute semantic class (the variable parameter). When method "initialize" is used, it would create an object for each attribute class in the training set, as stored in the classes configuration object.

Method "updateRequiredIterations" determines how datasets need to be iterated to compute existing features. Each feature has a required iteration type indicated by the "iterationType" property of the features group that contains it. In some cases, the computation of a feature requires iterating the instances of the input dataset in a top-down manner, while in other cases, the dataset has to be iterated bottom-up. Some features require a double iteration, from top to bottom and back, or viceversa. For example, feature "Depth in the dataset" requires a top-down iteration to be computed, since the value of an instance is computed from that of the upper instance plus one; however, the feature "Distance to the nearest attribute" requires a bottom-up iteration followed by a top-down one to be computed efficiently (the exact way in which

it is computed is not relevant, and there could be other implementations). Function "updateRequiredIterations" decides what are the optimal iterations required to compute all features and stores this information in class properties (ommited for brevity). For example, if a feature requires a bottom-up iteration and another feature require a top-down iteration followed by a bottom-up one, "updateRequiredIterations" will determine that the best course of action is to perform a top-down iteration followed by a bottom-up one, which covers both features.

Our framework implements the observer pattern to support incremental features, whose values are gradually updated. For example, dataset feature "number of instances" requires going over the dataset to count how many instances it contains. Instead of iterating the dataset only to compute this value, it can be updated while instances are iterated to compute other features. The main use of this pattern, however, is to compute statistical dataset features. These features consist in several statistical measures (such as the average, standard deviation, minimum, etc.) of each feature, for each semantic class. These would include, for example, the average number of digits among attributes that were labelled as a zipCode in the dataset, or the maximum number of letters among attributes that were labelled as an agency (note that these are hint-based features, since they require the presence of hints among instances. The framework, however, is oblivious to whether a feature is hint-based or hint-free). These and other dataset features are not useful for semantic labelling unless the user wants to label entire datasets. However, they can be useful for other purposes such as information verification, where entire datasets are verified using such features. For this reason, we decided to support the inclusion of such features in our framework.

Class "FeaturesTablesBuilder" takes care of storing the computed feature vectors in CSV files, separating attribute vectors from record vectors. This functionality, however, is optional, and when disabled, feature values will only be stored in memory.

Finally, method "run" computes the features of the current dataset (selected using method "setDataset"). Before computing features, TAPON-MT iterates over the attributes in the dataset to identify which ones are numeric and which ones are textual, so that features may take this information into account. Numerical attributes are detected using user-made patterns. If an attribute is detected to be numeric, its value is stored.

Example

We included the dataset in Figure 1 in a training set, and gave it as input the the features framework. The features of the attributes were computed as shown in Table 2. The ID column contains an id given to each instance. The class column contains the actual semantic class of the instance, which is known for training datasets. We only display some of the total 135 features, many of which correspond to large features groups that create a feature per semantic class. There were 39 semantic classes in the training set that included the example.

4.2.2. Models handler

The architecture of the models handler is presented in Figure 6. The models handler offers a high level interface to create models and use them to label datasets.

Class "ModelHandler" provides the methods used to label datasets. As we described, datasets are labelled in two stages: in the first one, only hint-free features are computed. In the second one, hint-based features are added. Because of this, there are two properties that correspond to both features calculators, which use different sets of features. The first one computes hint-free features, and the second one, hint-based ones. If no hint-based features are needed, the second features calculator will not compute any feature. Additionally, properties "tablesFolder" and "classifiersFolder" denote the folders where the tables with feature vectors and the trained classifiers will be stored respectively.

Method "trainModel" trains a model from a set of labelled datasets. The creation of the model takes place as explained in Figure 3. The features calculators are used to compute features. The model is a one-vs-all classifier where the aggregation of binary outputs is achieved by means of an additional multiclass classifier. Consequently, the classifiers stored by the model handler include: a binary classifier for each attribute semantic class, using only hint-free features; a multiclass classifier that takes the output of each of the former classifiers as features and outputs a final label as depicted in Figure 7; the same binary and multiclass classifiers, but using, additionally, hint-based features; and finally, the same hint-free and hint-based classifiers, but for records instead of attributes. For example, if there are 20 attribute semantic classes, and 10 record semantic classes, the model will be composed of $(20+1+20+1)+(10+1+10+1) = 64$ classifiers.

ID	Avg title similarity	Avg zipCode similarity	Node depth	Occurrences \b\p{Lu}+\b	Occurrences \d+[\\,]?\d*	Occurrences \p{N}	Occurrences \p{P}	Numeric value	Class
1	0.04	0.52	2.00	0.00	1.00	7.00	0.00	1750497.00	id
2	0.58	0.00	2.00	0.00	0.00	0.00	2.00	-1.00	title
3	0.25	0.00	2.00	1.00	0.00	0.00	0.00	-1.00	agency
4	0.05	0.53	2.00	0.00	3.00	8.00	2.00	-1.00	startDate
5	0.05	0.51	2.00	0.00	3.00	8.00	2.00	-1.00	expDate
6	0.04	0.51	2.00	0.00	1.00	5.00	0.00	50052.00	estimatedTotalAmt
7	0.47	0.00	3.00	0.00	0.00	0.00	0.00	-1.00	city
8	0.06	0.53	3.00	0.00	1.00	9.00	0.00	926173213.00	zipCode
9	0.50	0.00	3.00	0.00	0.00	0.00	1.00	-1.00	name

Table 2: Computed features.

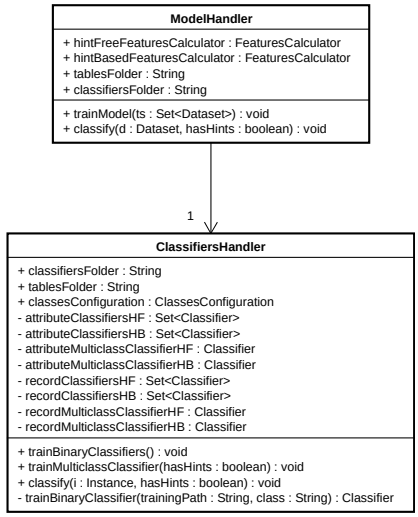


Figure 6: Class diagram of the models handler.

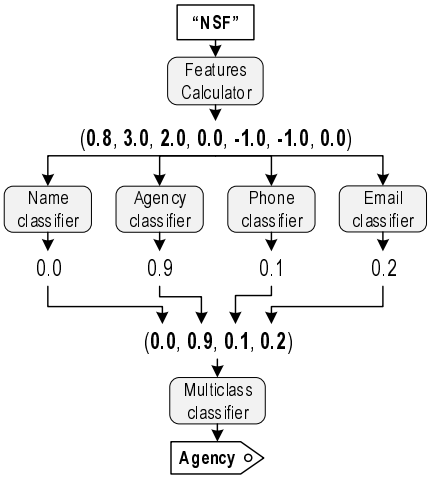


Figure 7: Instance classification.

The creation of a binary classifier per class may be a cause for concern when there is a large number of semantic classes (several hundreds), since the time required to train all of them could be large. However, data integration and similar tasks do not usually involve so many semantic classes, but only a few dozens. Pham et al. [21]'s approach, for example, also uses a one-vs-all classifier. However, if there is indeed a large number of classes, the following strategies can be used to decrease training time: tuning hyperparameters to increase training speed at the cost of classification accuracy; using a smaller set of features at the risk of losing useful ones; or using a single multiclass classifier instead of many binary ones, which may be problematic if the classifier does not handle well the large number of classes, and would require slight modifications to the architecture of TAPON-MT.

Note that we use a multiclass classifier to aggregate the binary scores instead of merely taking the class with the highest probability, which is the traditional approach in one-vs-all classifiers. We do this to properly deal with complex cases in which the correct class is not the one with the highest score. For example, let us suppose there are two classes, A and B. When an instance belongs to class A, the probabilities are <A=0.9, B=0.2>, and when it belongs to B, they are <A=0.9, B=0.8> (probabilities are independent, so they do not necessarily amount to one). In the second case, the probability of A is the highest one, which would lead a traditional one-vs-all classifier to choose it as the label. Our method, however, is more flexible, and would learn that an instance belongs to class B when the probability of B is high enough, independently of other, potentially higher probabilities. If our approach is not needed, TAPON-MT offers the possibility of just taking the class with

the highest probability.

Method "classify" applies a model (specified by the "classifiersFolder" property) to a dataset. If the dataset does not have hints, as indicated by the "hasHints" parameter, TAPON-MT applies the hint-free model, with its corresponding set of classifiers; if it does have them, it applies the hint-based model. For each instance, it first applies each binary classifier. This results in a set of classifier outputs (be it a purely binary result or a probability) that are used as the input of the multiclass classifier. The multiclass classifier outputs the final label of the instance, which may be used as a hint. If hint-based features are not needed, only the first iteration needs to be performed.

Class "ClassifiersHandler" provides low-level functionality related to the creation and use of classifiers. Its many properties correspond, mainly, to the classifiers of a model, separated into the 8 groups we described: for attributes and records, hint-free or hint-based, and binary or multiclass. Additionally, it stores a classes configuration, since creating multiclass classifiers require knowing what classes there are in the input datasets. Any classification technique can be used to implement both the binary and multiclass classifiers. For example, in our implementation of TAPON-MT, we train random-forest binary classifiers and neural network multiclass classifiers.

Method "trainBinaryClassifiers" trains a binary classifier for each semantic class by using method "trainBinaryClassifier", which trains each individual classifier and stores it in a given path.

Method "trainMulticlassClassifier" trains the multiclass classifier for both attributes and records, using the hint-free or hint-based classifiers depending on the value of the "hasHints" parameter.

Finally, method "classify" classifies a single instance, using hint-free or hint-based classifiers depending on the value of the "hasHints" parameter. The resulting label is stored in the instance. Information about the probability of the label is also included.

Example

An already trained model was applied to the dataset in Figure 1 by using it as input of the models handler. First, the framework classifies it specifying that there are no hints yet. Hint-free features are computed and the classifiers are applied. Most labels are correct, but some of them are wrong, as depicted in Figure 8. There are two wrong labels: the expDate attribute has been labelled as a startDate, and the performance record has been labelled as an awardee. The "classify" method is applied a second time, but now specifying that there are hints. After computing hint-based features, the hint-based classifiers are applied. The additional features help distinguish between similar cases: the expDate is near an attribute labelled as startDate, but has a higher value, while the startDate is near another attribute labelled as startDate, but has a lower value. The performance contains an attribute labelled as a location, while the awardee contains an attibute labelled as a name. After the second iteration, all labels are correct.

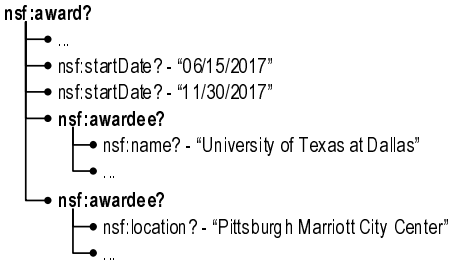


Figure 8: Hints after first iteration using the model handler.

5. Validation

This section describes our validation of TAPON-MT by means of an implementation with a web service and a graphical interface, as well as experimental results. The source code of TAPON-MT has been made available online[1].

5.1. Implementation

Our framework is supported by both a web service and a graphical interface.

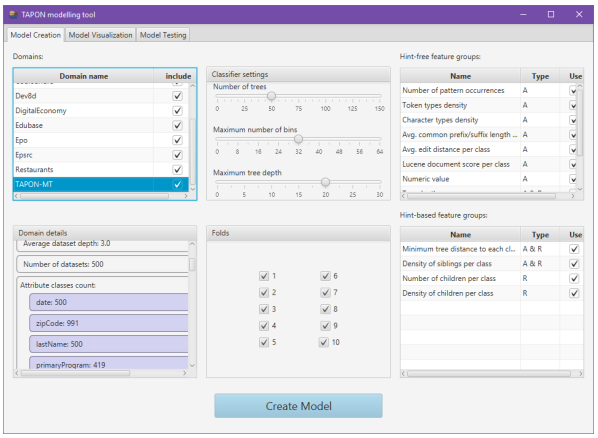
The web service was deployed using the Java Restlet library. It offers a "classify" POST method that takes a dataset in JSON format, and returns the same dataset with added fields that correspond to the final label of each instance according to the multiclass classifier, as well as the top 4 most likely labels according to the binary classifiers' confidence scores. Figure 9 depicts the example in Figure 9 in JSON format, without labels.

The service requires the existence of already trained models, which can be obtained with creation scripts or our graphical interface.

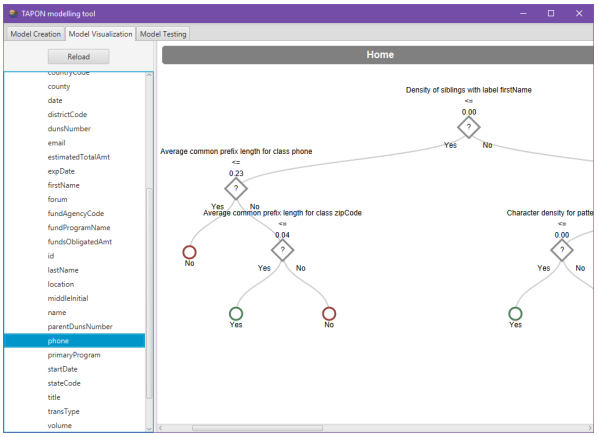
[1] https://github.com/dayala1/TAPON-MT

```
{"children": [
  {"textualValue": "Partial Support for..."},
  {"textualValue": "NSF"},
  {"textualValue": "15000"},
  {"textualValue": "06/15/2017"},
  {"textualValue": "11/30/2017"},
  {"children": [
    {"textualValue": "Walter Hu"}
  ]},
  {"children": [
    {"textualValue": "University of Texas..."},
    {"textualValue": "Richardson"},
    {"textualValue": "750803021"}
  ]},
  {"children": [
    {"textualValue": "Pittsburgh Marriott..."},
    {"textualValue": "Pittsburgh"},
    {"textualValue": "152302999"}
  ]}
]}
```

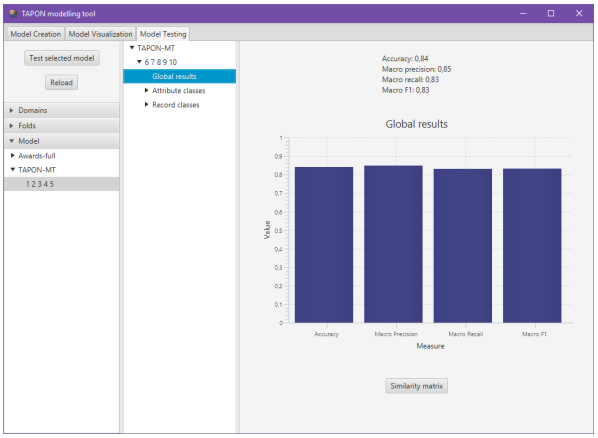
Figure 9: Dataset in JSON format.



(a) Model creation.



(b) Model visualisation.



(c) Model testing.

Figure 10: TAPON-MT graphical interface.

The graphical interface has been implemented using Java 1.8.0_151. We have used re2j 1.1 to implement regular expressions, Lucene 5.4.1 to create indexes, and Spark 2.2.0 to implement classification techniques which are the basis of our models.

It has three modules, namely: Model creation, Model visualisation, and Model testing.

In the Model creation module (cf. Figure 10(a)) the user can create models from a training set. The training files (in JSON format) can be split into folders that correspond to different domains to separate those from different sources. Each domain folder must in turn be split into ten fold folders. The user can then select what domains and folds will be used to train the model. The files of a domain folder can be analysed to obtain some metadata, such as the number of classes, average dataset depth or the number of attributes. These details are stored in a JSON file and can be visualised. Other applications can create or modify this file to add further information about a domain. The user can also select what features groups will be used by the model, as well as the classifier settings (which, in our implementation, are related to random forest classifiers).

The Model visualisation module (cf. Figure 10(b)) allows the user to visualise the random forest classifiers that shape a model. Since a model consists of a classifier per class, the user can select a model (the training domains and folds), and visualise every decision tree in the random forest associated to every class (this visualisation is specific to decision trees, so other classification techniques would require different visualisations). This allows the user to check that the classifiers make sense, and that certain features are taken into account. For example, if a model yields bad results, it could be because the classifier is not using all the available features for some reason (e.g. the maximum

decision tree depth is set to a low value). This can be done for both the hint-free and the hint-based classifiers. We used this visualisation to check that hint-based features are actually useful when introduced, mainly in record classifiers.

Finally, the Model testing module (cf. Figure 10(c)) allows the user to choose a previously created model and then apply it to a subset of the labelled information. The user must select the model to test (the domains and folds used for training), as well as the domains and folds used for testing. Note that the user can test a model with datasets from domains that contain classes that are unknown to the model. This can be used to test how the model behaves when facing information that does not belong to any known class. The results of the test include information related to each individual class, as well as global results. The class results include the precision, recall, and F_1 score. The global results include the accuracy, macro precision, macro recall, and macro F_1 score, as well as a classes similarity matrix.

The classes similarity matrix is a technique to asses how similar pair of classes are from the point of view of the classifier. Figure 11 depicts a similarity matrix corresponding to 39 classes using a color scale ranging from 0.0 (dark purple) to 1.0 (yellow) and hidden class names (visible by hovering over each square, since some class names are long). The similarity of two classes is computed as the cosine similarity between their corresponding rows in the confusion matrix. The similarity matrix is, ideally, the identity matrix: every class is perfectly similar to itself (which is always the case), and completely made apart from the others. High values for different classes mean that the classes have been classified in a similar way, and therefore the model has not been able to separate them. Note that the similarity is not an objective measurement of how similar two classes are, since it depends on the classifier, and therefore classes that are highly similar using some classification techniques and features can be perfectly separated using other classification techniques and features. The similarity matrix is useful to identify the cases in which a particular model fails. For instance, in the matrix example we provided, we observed that the pairs of classes with high similarity were almost undistinguishable, even for a human, so the accuracy of the classifier, even if not perfect, was hard to surpass.

Figure 11: Similarity matrix example.

5.2. Experimental results

Apart from the main contributions of our framework, as described in Section 2, we have performed some tests to check that the models it creates work in practice and is able to model information successfully. Our experiments consists of 10-fold cross-validation applied to a subset of the National Science Foundation Awards dataset. The subset consists of the first 500 most recent ended awards, from those that ended in 2017. These have a total of 2223 record instances distributed among 5 record classes and 17723 attribute instances (29.48% of them numerical) distributed among 35 attribute classes. Datasets have an average depth of 3 (1 being a flat record), and records contain, on average, 38.89 instances. These datasets were chosen for their non-flat structures, the high number of classes, and the variety of attributes, which include numerical values. We performed our experiments on a computer with an Intel Xeon E7-4807 that ran at 1.87 GHz, had 16 GB of RAM, Windows 7 Pro 64-bit, Java 1.7, and Spark 2.10 for Java. No changes were made to their default configurations.

We compare the results obtained by models created with our framework, TAPON-MT without hints (denoted by "TAPON-MT") and with hints (denoted by "TAPON-MT(H)"), to those of an implementation of RAPTURE [11] that uses random forest classifiers with the proposed features, an implementation of Ramnandan et al. [22]'s proposal (denoted by "RAM"), and an implementation of Pham et al. [21]'s proposal. We have selected these

Tool	Precision	Recall	F1	Accuracy
TAPON-MT(H)	[0.85, 0.87]	[0.83, 0.85]	[0.84, 0.86]	[0.84, 0.85]
TAPON-MT	[0.90, 0.91]	[0.73, 0.75]	[0.80, 0.82]	[0.71, 0.74]
RAPTURE	[0.76, 0.78]	[0.62, 0.64]	[0.69, 0.70]	[0.62, 0.64]
RAM	[0.76, 0.77]	[0.60, 0.63]	[0.67, 0.70]	[0.61, 0.63]
PHAM	[0.81, 0.83]	[0.54, 0.57]	[0.65, 0.67]	[0.54, 0.57]

Table 3: Results summary displaying the 95% confidence interval.

since, as we described in Section 2, most proposals have limitations that make them completely inapplicable in our set-up (such as only being able to label tables or named entities). Since both are only able to label attributes, their quality measures are computed considering only the attributes in the datasets (though their labelling is only partial, they are not inapplicable). All classifiers were created using their Spark 2.20 Java implementation. Random forest classifiers were created with 50 trees and a maximum depth of 15 (other settings were set to their defaults). The neural networks of our multiclass classifiers had a single dense layer (other settings were set to their defaults).

Results are shown in Figure 12. The measure that better describes the quality of the results is the accuracy, since it represents the fraction of correct labels. Note that our main contribution is the flexibility of our framework as described in Section 2, independently of the results obtained a specific set of features or datasets. TAPON-MT achieves a good accuracy, even when labelling both records and attributes in an individual way. While hint-based features improve accuracy in a significant way, the model without them also achieves good results thanks to the large features catalogue. Table 3 contains a summary of the results. TAPON-MT(H) labels correctly 85% of instances, while the results by other proposals range from 55% to 65%. Note that there are 40 classes, so a random classifier would label instances correctly 2.5% of times.

Across the 10 experiments, the average training time of TAPON-MT(H) was 36.59 minutes, with a standard deviation of 1.49 minutes. This time seems to be reasonable, since the model only has to be trained once. The average application time per instance was 0.11 seconds, which is enough for labelling datasets in real time.

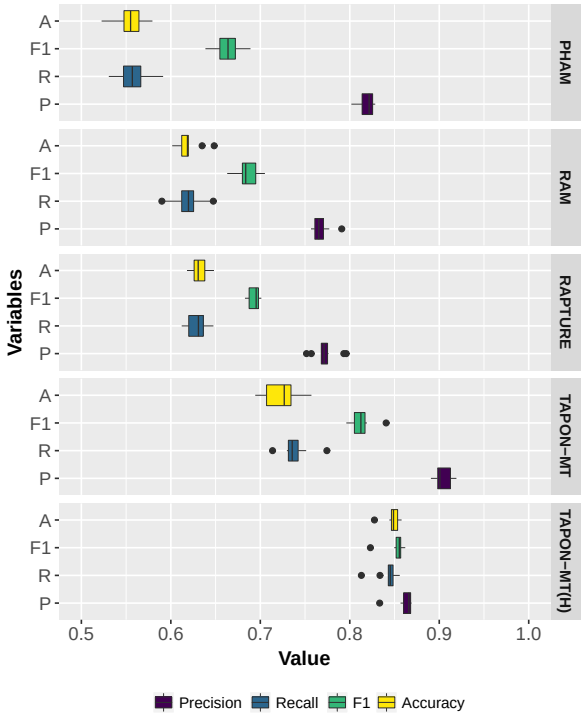


Figure 12: Experimental results.

6. Conclusions

Semantic labelling is an area of research that has become more relevant than ever in the recent years due to the ever-increasing presence of available structured data sources in the Web. Unfortunately, most of the existing approaches to label structured information have been devised with specific situations in mind, and thus do not offer a more complete solution due to their restrictions, namely: they only accept some structures as input, which greatly limits their applicability to only some sources of data; they are unable to label both records and attributes, which prevents them from fully labelling datasets; they are tied to certain features or techniques, without offering customisable models that can be expanded with new techniques or features that are suitable to specific cases; they assume that instances are labelled in groups that share the same class, having their performance hindered when such groups are not available and datasets are labelled individually; they are tied to entity matching, thus being unable to label records or attributes that do not correspond to named entities; they do not detect numeric data, which provides additional information that can be crucial to

the model; finally, they do not support complex features.

In this article, we present TAPON-MT, a framework to perform semantic labelling based on machine learning techniques, with a features calculation and modelling architecture that does not present any of the aforementioned problems. It deals with both records and attributes from generic structures; it is highly customisable, being able to dynamically generate complex features using features groups; it labels instances in an individual basis without assuming there are groups; it does not require the presence of named entities or performing any kind of entity matching; it detects numeric data so that additional information can be used to compute features; and it supports complex features: similarity features (by storing examples from the training set), dynamically generated features (by using features groups), incremental features (by using the listener pattern), and hint-based features (by means of a two-phase workflow).

We have implemented our framework to show its usability when creating models and testing them. We have also validated it using real-world data from the National Science Foundation RDF dataset, which includes a large number of varied classes in deep structures. Our validation results show that TAPON-MT is effective and efficient in practice, achieving accuracy results that are, at least, on par with other proposals even when having to label records in addition to attributes. These results suggest that our framework is promising enough for real-world semantic labelling scenarios.

Acknowledgements

Our work was supported in the Spanish R&D&I programme by grant TIN2016-75394-R.

References

References

[1] Daniel Ayala, Inma Hernández, David Ruiz, and Miguel Toro. Tapon: A two-phase machine learning approach for semantic labelling, 2019.

[2] Michele Banko, Michael J. Cafarella, Stephen Soderland, Matthew Broadhead, and Oren Etzioni. Open information extraction from the web. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 2670–2676, 2007. URL `http://ijcai.org/Proceedings/07/Papers/429.pdf`.

[3] Philip A. Bernstein and Laura M. Haas. Information integration in the enterprise. *Commun. ACM*, 51(9):72–79, 2008. doi: 10.1145/1378727.1378745. URL `http://doi.acm.org/10.1145/1378727.1378745`.

[4] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst.*, 5(3):1–22, 2009. URL `http://dx.doi.org/10.4018/jswis.2009081901`.

[5] Valter Crescenzi, Giansalvatore Mecca, and Paolo Merialdo. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, pages 109–118, 2001. URL `http://www.vldb.org/conf/2001/P109.pdf`.

[6] Jérôme Euzenat and Pavel Shvaiko. *Ontology Matching, Second Edition*. Springer, 2013. ISBN 978-3-642-38720-3.

[7] National Science Foundation. NSF Awards API specification. `https://www.research.gov/common/webapi/awardapisearch-v1.htm`. Accessed: 2018-06-08.

[8] Inma Hernández, Carlos R Rivero, and David Ruiz. Deep web crawling: a survey. *World Wide Web*, pages 1–34, 2018.

[9] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160:3–24, 2007.

[10] Nicholas Kushmerick. Regression testing for wrapper maintenance. In *AAAI/IAAI*, pages 74–79, 1999. URL `http://www.aaai.org/Library/AAAI/1999/aaai99-011.php`.

[11] Nicholas Kushmerick. Wrapper verification. *WWW*, 3 (2):79–94, 2000. doi: 10.1023/A:1019229612909.

[12] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 233–246, 2002. doi: 10.1145/543613.543644. URL `http://doi.acm.org/10.1145/543613.543644`.

[13] Kristina Lerman, Steven Minton, and Craig A. Knoblock. Wrapper maintenance: A machine learning approach. *J. Artif. Intell. Res.*, 18:149–181, 2003. doi: 10.1613/jair.1145.

[14] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *PVLDB*, 3 (1):1338–1347, 2010. URL `http://www.comp.nus.edu.sg/~vldb2010/proceedings/files/papers/R118.pdf`.

[15] Robert McCann, Bedoor K. AlShebli, Quoc Le, Hoa Nguyen, Long Vu, and AnHai Doan. Mapping maintenance for data integration systems. In *VLDB*, pages 1018–1030, 2005. URL `http://www.vldb2005.org/program/paper/fri/p1018-mccann.pdf`.

[16] Hannes Mühleisen and Christian Bizer. Web data commons - extracting structured data from two large web corpora. In *LDOW*, 2012. URL `http://ceur-ws.org/Vol-937/ldow2012-inv-paper-2.pdf`.

[17] Varish Mulwad, Tim Finin, and Anupam Joshi. Semantic message passing for generating linked data from tables. In *ISWC*, pages 363–378, 2013. doi: 10.1007/978-3-642-41335-3_23.

[18] Sebastian Neumaier, Jürgen Umbrich, Josiane Xavier Parreira, and Axel Polleres. Multi-level semantic labelling of numerical values. In *International Semantic Web Conference (1)*, pages 428–445, 2016. URL `http://dx.doi.org/10.1007/978-3-319-46523-4_26`.

[19] Natalya Fridman Noy. Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33 (4):65–70, 2004. doi: 10.1145/1041410.1041421. URL http://doi.acm.org/10.1145/1041410.1041421.

[20] Tugba Özacar. A tool for producing structured interoperable data from product features on the web. *Inf. Syst.*, 56:36–54, 2016. URL http://dx.doi.org/10.1016/j.is.2015.09.002.

[21] Minh Pham, Suresh Alse, Craig A. Knoblock, and Pedro A. Szekely. Semantic labeling: A domain-independent approach. In *International Semantic Web Conference (1)*, pages 446–462, 2016. URL http://dx.doi.org/10.1007/978-3-319-46523-4_27.

[22] S. K. Ramnandan, Amol Mittal, Craig A. Knoblock, and Pedro A. Szekely. Assigning semantic labels to data sources. In *ESWC*, pages 403–417, 2015. doi: 10.1007/978-3-319-18818-8_25.

[23] Dominique Ritze, Oliver Lehmberg, and Christian Bizer. Matching html tables to dbpedia. In *Proceedings of the 5th International Conference on Web Intelligence, Mining and Semantics*, page 10. ACM, 2015.

[24] Petros Venetis, Alon Y. Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the Web. *PVLDB*, 4(9):528–538, 2011. URL http://www.vldb.org/pvldb/vol4/p528-venetis.pdf.

[25] Ziqi Zhang. Effective and efficient semantic table interpretation using tableminer+. *Semantic Web*, (Preprint):1–37, 2016.