Combining Lexical and Semantic Similarity Measures with Machine Learning Approach for Ontology and Schema Matching Problem

© Lev Bulygin
Lomonosov Moscow State University,
Moscow, Russia
buliginleo@yandex.ru

Abstract. Ontology and schema matching is one of the most important tasks for data integration. We suggest to combine the different matchers with machine learning approach. The features are the outputs of lexical and semantic similarity functions. Naive Bayesian classifier, logistic regression and gradient tree boosting have been trained on these features. The proposed approach is tested on the OAEI 2017 benchmark "conference" with the various splits of the data on train and test sets. Experiments show that final combined model in element-level matching outperformed the single matchers. Results are compared with EditDistance matcher and WordNet matcher.

Keywords: ontology matching, element-level matcher, lexical matcher, semantic matcher, word2vec, word2net.

1 Introduction

Data integration is now a very important problem, as the amount of data is growing very much. One of the most important steps in automatic data integration is the comparison of ontologies or schemas. This problem is traditionally solved manually or semi-automatically. Manual ontology or schema matching is very labor-intensive, long in time and prone to errors. Therefore now the task is to maximally automate the process of comparing the circuit with the greatest accuracy.

For schema matching and ontology matching element-level and structure-level matchers are used. Element-level matchers use information of elements of schema (name of columns, description). Structure-level matchers use information of structure (type similarity, key properties, hierarchy of elements).

Recently, element-level matchers have been widely used lexical and semantic information. For getting semantic information WordNet and Word2vec are used. Word2vec allow us to get a meaning of the word and we can use this for improving ontology and schema matching. So we can combine many lexical and semantic information into one similarity matrix and train a machine learning model for trying to solve ontology and schema matching problems.

This work is performed as Master thesis. The main goal of this work is to propose a approach for solving ontology and schema matching problem with the greatest accuracy. To achieve this goal we need to perform the following tasks: review related work, create the architecture of solution, select of information for matching and create experiments. We compared our approach on OAEI 2017 "conference" ontologies with single matchers: edit distance and WordNet.

In Section 2 we reviewed related work and it's evolution. Section 3 describes the setup of the paper. In Section 4 presents our matching algorithm in detail. Section 5 shows the experiments and the analysis. We conclude this paper in Section 6.

2 Related Work

Schema matching and ontology matching are very similar. In this article, there is no difference for schema matching and ontology matching because we only work with the names of the entities. So we considered the works about schema matching and ontology matching together. In [1] a review of approaches for automatic schema matching are conducted. They introduced the classification of matchers and concluded that it is impossible to create a fully automatic matching system, universal for all subject areas. One of the most important conclusions is that hybrid matching system is better than single system. Intuitively, this is obvious, since a hybrid matcher uses more information to make a decision.

In [2] the ready-made solutions for automatic schema matching are compared and the concept of pre-match effort and post-match effort introduced. Pre-match effort is learning of the matcher, configuration of thresholds and weights. Post-match effort is correction and improvement of the match output. In the same year, in [3] the authors described their COMA system. The authors introduced new methods of post-match effort: reuse of matchings, aggregation of individual matchers. In [4] the Target-based Integration Query System (TIQS) are described. In [5] a new classification on matching dimensions are presented. A natural issue is uncertainty, In [6] uncertainty are considered, monotonicity and statistical monotonicity are introduced . All the above articles don't use machine learning for aggregation

Proceedings of the XX International Conference "Data Analytics and Management in Data Intensive Domains" (DAMDID/RCDL'2018), Moscow, Russia, October 9-12, 2018

results of matchers.

In [7] Bayesian Networks are used for combining ontology matching. As the features lexical information (N-gram, Levenshtein, Dice Coefficient etc) were used. The best accuracy is 88%.

In [8] the outputs from several single matchers are used and the authors trained on this data a multi-layer neural network.The authors used lexical information and the WordNet similarity.

In [9] ontology matching problem in detail are described: applications, classifications of ontology matching techniques, evaluations and alignment. They suggest three dimensions of building correspondences: conceptual, extensional and semantic.

In [10] Multiple Concept Similarity for ontology mapping are described. The author reduced the ontology mapping problem to the machine learning problem. They used lexical information (prefix, suffix, Edit distance, n-gram), semantic information (WordNet) and special types of similarity called Word List similarity (a similarity for sentences).

In [11] a machine-learning approach to ontology alignment problem are described. The authors used lexical information, semantic information (WordNet) and structural information for training Support Vector Machine, K-Nearest Neighbours, Decision Tree, AdaBoost models. The authors improves F-measure criterion up to 99%.

In [12] Bayeasian Networks are used for composition of matchers. The resulting model achieved 81% accuracy. The authors used the outputs of the lexical matchers, structure-level matchers, synonym matchers and instance-level matchers.

In [13] all stages of ontology matching problem in detail are described: feature selection, methods of combining matchers and experiments.

In [14] word2vec embeddings for ontology matching problem are used. Authors proposed the new algorithm of matching and sentence2vec algorithm. The results matcher are compared with various types of WordNet, EditDistance matchers. In [15] the combination of word2vec features and a neural network are used. One of the most important problems is that articles using machine learning can not be compared with each other because of the lack of a unified benchmark. The OAEI is not intended to use machine learning, so the training dataset is chosen by the author of the article.

At the moment, the most promising option is the using of machine learning to create a hybrid model with lexical, semantic and structure information. In this article, we want to connect many different single matchers to one hybrid matcher using machine learning. One of the newest features we used is Word2vec. So far we only used the single element-level matchers.

All reviewed papers use widely the lexical information and WordNet distances. In this paper we used 29 similarity metrics from 17 various single element-level matchers.

3 Problem Statement, Evaluation Metrics and Similarity Measures

3.1 Problem Statement

An ontology *O* is by a 3-tuple *(C, P, I)*. *C* is the classes, denoting the concepts. *P* is the relations within the ontology. *I* is the instances of classes. The task of *ontology matching* is to find the alignment between entities in a source ontology O_1 and a target ontology O_2.

An alignment is a set $\{(e_1, e_2, con) | e_1 \in O_1, e_2 \in O_2\}$, where e_1 is an entity in O_1, e_2 is an entity in O_2, and con is the confidence of the correspondence.

Our algorithm works only with names of entities without the structure information. The input of our algorithm is the two ontologies: source and target. The output of our algorithm is the alignment.

3.2 Evaluation Metrics

The effectiveness of ontology matching could measured by precision, recall and F-measure.

$$precision = \frac{|A \cap B|}{|A|}, \quad (1)$$

$$recall = \frac{|A \cap B|}{|B|}, \quad (2)$$

$$F - measure = \frac{2 \cdot precision \cdot recall}{precision + recall}, \quad (3)$$

where *A* is a predict alignment and *B* is a true alignment.

3.3 Used Similarity Measures

As features for machine learning we used the similarity measures. We extract from ontology the names of entities. Further we need to create *sets* of words from names for computing some similarity measures.

Example. We have two names of entities: “early_paid_applicant” and “early-registered-participant”. The *sets* are [“early”, “paid”, “applicant”] and [“early”, “registered”, “participant”].

Metrics based on lexical information from names of entities:

- *N-gram* [11]. Let *ngram(S, N)* be the set of substrings of string *S* of length *N*. The n-gram similarity for two strings *S* and *T*:

$$ngram(S,T) = \frac{|ngram(S,N) \cap ngram(T,N)|}{min(|S|,|T|) - N + 1} \quad (4)$$

- *Dice coefficient* [11]. The Dice similarity score is defined as twice the shared information (intersection) divided by sum of cardinalities. For two sets *X* and *Y*, the Dice similarity score is:

$$dice(X,Y) = \frac{2 * |X \cap Y|}{|X| + |Y|} \quad (5)$$

- *Jaccard similarity* [11]. For two sets *X* and *Y*, the Jaccard similarity score is:

$$jaccard(X,Y) = \frac{|X \cap Y|}{|X \cup Y|} \quad (6)$$

- *Jaro measure* [11]. The Jaro measure is a type of edit distance, developed mainly to compare short strings, such as first and last names.

- *Substring similarity* [11]. For longest substring T of two strings X and Y substring similarity is:

$$substring(X,Y) = \frac{2*|T|}{|X|+|Y|} \quad (7)$$

- *Monge-Elkan* [11]. Token-based and internal similarity function for tokens:

$$sim_{ME}(x,y) = \frac{1}{|x|}\sum_{i=1}^{|x|} max_{j=1,|y|} sim'(x[i],y[j]) \quad (8)$$

- *Smith-Waterman* [11]. The Smith-Waterman algorithm performs local sequence alignment; that is, for determining similar regions between two strings.
- *Needleman-Wunsch* [11]. The Needleman-Wunsch distance generalizes the Levenshtein distance and considers global alignment between two strings. Specifically, it is computed by assigning a score to each alignment between the two input strings and choosing the score of the best alignment, that is, the maximal score.
- *Affine gap* [17]. Returns the affine gap score between two strings. The affine gap measure is an extension of the Needleman-Wunsch measure that handles the longer gaps more gracefully.
- *Bag distance.* The number of common symbols in two strings.
- *Cosine similarity* [11]. For two sets *X* and *Y*:

$$cosine(X,Y) = \frac{|X \cap Y|}{\sqrt{|X| \cdot |Y|}} \quad (9)$$

- *Partial Ratio.* Given two strings X and Y, let the shorter string (X) be of length m. It finds the fuzzy wuzzy ratio similarity measure between the shorter string and every substring of length m of the longer string, and returns the maximum of those similarity measures.
- *Soft TFIDF [21]*. A variant of TF-IDF that considers words equal based on Jaro Winkler rather than exact match.
- *Editex.* Editex is a phonetic distance measure that combines the properties of edit distances with the letter-grouping strategy used by Soundex and Phonix.
- *Generalized Jaccard.* This similarity measure is softened version of the Jaccard measure. The Jaccard measure is promising candidate for tokens which exactly match across the sets.
- *JaroWinkler.* The Jaro-Winkler measure is designed to capture cases where two strings have a low Jaro score, but share a prefix and thus are likely to match.
- *Levenshtein distance* [11]. Levenshtein distance computes the minimum cost of transforming one string into the other.
- *Partial Token Sort.* For two strings *X* and *Y*, the score is obtained by splitting the two strings into tokens and then sorting the tokens. The score is then the fuzzy wuzzy partial ratio raw score of the transformed strings.
- *Ratio.* For two strings *X*, *Y*:

$$Ratio = round(2.0 \cdot \frac{M}{T} \cdot 100), \quad (10)$$

- where *T* is the total number of characters in both strings, and *M* is the number of matches in the two strings *X* and *Y*.
- *Soundex* [6]. Phonetic measure such as soundex match string based on their sound. These measures have been especially effective in matching names, since names are often spelled in different ways that sound the same.
- *Token Sort.* Fuzzy Wuzzy token sort ratio raw raw_score is a measure of the strings similarity as an int in the range [0, 100]. For two strings X and Y, the score is obtained by splitting the two strings into tokens and then sorting the tokens. The score is then the fuzzy wuzzy ratio raw score of the transformed strings.
- *Tversky Index.* For sets *X* and *Y*:

$$TI(X,Y) = \frac{|X \cap Y|}{|X \cap Y| + \alpha|X-Y| + \eta|Y-X|}, \quad (11)$$

- where $\alpha, \eta > 0$.
- *Overlap coefficient.* For sets *X* and *Y*:

$$OC(X,Y) = \frac{|X \cap Y|}{min(|X|,|Y|)} \quad (12)$$

Metrics based on semantic information from names of entities:

- *WordNet similarity.* Best score of similarity between synonyms of one word with synonyms of other word. We use a sentence similarity algorithm from [14].
- *Word2vec and Sentence2vec similarity.* Word2vec is a model that are used to produce word embeddings. We used word2vec model trained by the texts of Google News. The dimensionality of these vectors is 300. We used sentence2vec algorithm from [14].

4 Proposed Matching Algorithm

The input of the matching system is the two ontologies. For each ontology the system extracts the names of the entities. Further it generate all possible pairs of the names of the ontology with the names of the other ontology. For all pairs system computes the all similarity measures. All outputs of the similarity measures are concatenated into the one similarity matrix. The computed features are used as input to a machine learning model.

Program 1 Element-level matching algorithm

```
Input: Ontology1, Ontology2 - input ontologies
or schemas, THRESHOLD - threshold for create
matching between elements
Output: alignment - output alignment for
Ontology1 and Ontology2
for Entity1 ∈ Ontology1 do
  for Entity2 ∈ Ontology2 do
    Name1 ← get_name(Entity1)
    Name2 ← get_name(Entity2)
    Features ← get_features(Name1, Name2)
```

```
    Match ← predict_match(Features)
    if Match > THRESHOLD then
      alignment.append((Name1,Name2))
    end if
  end for
end for
return alignment
```

Program 2 Creating dataset and training a machine learning model

```
Input: TrueAlignments - set of true alignments,
TrainAlignments - set of train pairs

Output: Model - trained model for predict
matching

for Ontology1, Ontology2 in TrueAlignments do
  for Entity1 ∈ Ontology1 do
    for Entity2 ∈ Ontology2 do
      if (Entity1, Entity2) ∈ TrainAlignments
      then
        add_to_train(Entity1, Entity2, 1)
      else
        add_to_train(Entity1, Entity2, 0)
      end if
    end for
  end for
end for
Model.train()
return Model
```

For training a machine learning model we need the ontologies and the alignments. We split the alignments on the training alignments and testing alignments. Further we train a machine learning model on the features from train alignments. Then we can evaluate F-measure on testing alignment.

5 Preliminaries and Results

5.1 Collection of Data

The experiments were conducted at the dataset "conference" from OAEI 2017. The dataset consists 16 ontologies from the same domain (conference organisation) and 21 true ontology matchings. This dataset is convenient for machine learning because all ontologies are from the same domain and this dataset has the several true matchings between ontologies.

We get entities from ontologies with parsing code on Python and get array of tuples: *(entity1, entity2, match)*, where *entity1*, *entity2* - string names of entities, *match* - bool variable, 1 means that the entities are matched, 0 means that the entities are not matched. We generated 29 similarity measures from 3.3 with Python packages: py_stringmatching[24], fuzzycomp[25], gensim[26].

The final dataset is very unbalanced: 391 572 negative samples and 305 positive samples. After generating features we split data randomly on train and test datasets in equal proportions. In Table 1 are described our split.

Table 1 Our split of dataset "conference" from OAEI 2017 benchmark

Train pairs	Test pairs
Iasted, Sigkdd Cmt, Confof Confof, Edas Edas, Iasted Confof, Ekaw Cmt, Iasted Edas, Sigkdd Confof, Sigkdd Conference, Confof Cmt, Edas	Conference, Ekaw Ekaw, Sigkdd Cmt, Sigkdd Cmt, Ekaw Ekaw, Iasted Edas, Ekaw Conference, Iasted Conference, Sigkdd Conference, Edas Confof, Iasted Cmt, Conference

5.2 Experiments

As a baseline solution we take EditDistance and WordNet matchers. For every matcher we select threshold with the best F-measure on the test dataset. As the machine learning models we chose Naive Bayes Classifier[27], Logistic regression[28] as the simple models and Gradient boosting[29] as the complex model.

Table 2 The results on test dataset

Method	Precision, %	Recall, %	F-measure, %
EditDistance	50	35.8	41.7
WordNet	9.1	38.2	14.7
Naive Bayes Classifier	2.13	80.08	4.15
Logistic regression	75.58	40.12	52.41
XGBoost	69.15	45.67	**55.01**

We can see on Table 2 that XGBoost is the model with best F-measure 55.01%. The worst model is Naive Bayes Classifiers with F-measure 4.15%. EditDistance is better than WordNet by 27%. For getting more stable results we splitted the datasets randomly on training and testing pairs 20 times and averaged the results. The results are show on Table 3. We can see that XGBoost are remained the best model.

[24] https://github.com/kvpradap/py_stringmatching
[25] https://github.com/fuzzycode/fuzzycomp
[26] https://github.com/RaRe-Technologies/gensim
[27] http://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html
[28] http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
[29] https://github.com/dmlc/xgboost

Table 3 The averaged results on 20 various random splits on train and test datasets

Method	Precision, %	Recall, %	F-measure, %
EditDistance	47.31	39.2	42.82
WordNet	11.87	41.69	18.35
Naive Bayes Classifier	2.63	81.49	5.10
Logistic regression	74.98	43.43	54.95
XGBoost	72.72	47.51	**57.29**

6 Conclusions

In this paper, we combined the lexical and semantic similarity measures into one hybrid model. The experiments show that hybrid model is better than single matchers.

In future work we want to run our solution on other benchmarks. We must add the structure-level and instance-level features because in our dataset there are examples in which the names of entities are exactly the same but they are not matched.

Acknowledgments. This work is supervised by Sergey Stupnikov, Institute of Informatics Problems, Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences".

References

[1] Rahm, E., Bernstein, A.: A survey of approaches to automatic schema matching. In: The International Journal on Very Large Data Bases, vol. 10, issue 4, pp. 334-350 (2001). doi: 10.1007/s007780100057

[2] Do, H., Melnik, S., Rahm, E.: Comparison of Schema Matching Evaluations. In: Revised Papers from the NODe 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems, pp. 221-237 (2002). doi: 10.1007/3-540-36560-5_17

[3] Do, H., Rahm, E.: COMA: a system for flexible combination of schema matching approaches. In: VLDB '02 Proceedings of the 28th international conference on Very Large Data Bases, pp. 610-621 (2002). doi: 10.1016/B978-155860869-6/50060-3

[4] L. Xu, D. Embley.: Automating Schema Mapping for Data Integration. (2003). http://www.deg.byu.edu/papers/AutomatingSchemaMatching.journal.pdf

[5] Shvaiko, P., Euzenat, J.: A Survey of Schema-Based Matching Approaches. In: Journal on Data Semantics IV, pp. 146-171 (2005). doi: 10.1007/11603412_5

[6] Gal, A.: Why is Schema Matching Tough and What Can We Do About It? In: ACM SIGMOD Record, vol. 35, issue 4, pp. 2-5 (2006). doi: 10.1145/1228268.1228269

[7] Svab, O., Svatek, V.: Combining Ontology Mapping Methods Using Bayesian Networks. In: OM'06 Proceedings of the 1st International Conference on Ontology Matching, vol. 225, pp. 206-210.

[8] Hariri, B., Abolhassani, H., Sayyadi, H.: A Neural-Networks-Based Approach for Ontology Alignment. (2006). https://www.jstage.jst.go.jp/article/softscis/2006/0/2006_0_1248/_article

[9] Euzenat, J., Shvaiko, P.: Ontology Matching. Springer-Verlag Berlin Heidelberg, Berlin (2007). doi: 10.1007/978-3-642-38721-0

[10] Ichise, R.: Machine Learning Approach for Ontology Mapping Using Multiple Concept Similarity Measures. In: Seventh IEEE/ACIS International Conference on Computer and Information Science, (2008). doi: 10.1109/ICIS.2008.51

[11] Nezhadi, A., Shadgar, B., Osareh, A.: Ontology Alignment Using Machine Learning Techniques. In: International Journal of Computer Science & Information Technology, vol. 3, pp. 139-150 (2011). doi: 10.5121/ijcsit.2011.3210

[12] Nikovsi, D., Esenther, A., Ye, X., Shiba, M., Takayama, S.: Bayesian Networks for Matcher Composition in Automatic Schema Matching. In: Mitsubishi Electric Research Laboratories (2011). doi: 10.1.1.644.2168

[13] Ngo, D.: Enhancing Ontology Matching by Using Machine Learning, Graph Matching and Information Retrieval Techniques. In: University Montpellier II - Sciences et Techniques du Languedoc (2012). doi: 10.1.1.302.587

[14] Zhang, Y., Wang, X., Lai, S., He, S., Liu, K., Zhao, J., Lv, X.: Ontology Matching with Word Embeddings. In: Chinese Computational Linguistics and Natural Language Processing Based on Naturally Annotated Big Data, pp 34-45 (2014). doi: 10.1007/978-3-319-12277-9_4

[15] Jayawardana, V., Lakmal, D., Silva, N., Perera, A., Sugathadasa, K., Ayesha, B.: Deriving a Representative Vefctor for Ontology Classes with Instance Word Vector Embeddings. In: 2017 Seventh International Conference on Innovative Computing Technology (INTECH), (2017). doi: 10.1109/INTECH.2017.8102426