Meta-Learning Framework for Schema Matching

Elizabeth Witten

December 2022

Abstract

Given the large and heterogeneous nature of data lakes, dataset discovery is a difficult task on its own. Schema matching—the task of finding matching pairs of columns between a source and target schema—is an important component to finding relationships between two tables. In this paper, a novel meta-learned schema matching technique is introduced and evaluated against other instance-based schema matching techniques. A detailed look at the results offers insight into where meta-learning and data augmentation can improve matching effectiveness.

1 Introduction

Schema matching is a crucial component in almost all data science tasks. Data lakes can store very large amounts of heterogeneous data, and related data sources are usually not linked together. This makes data integration and dataset discovery difficult. Ideally, a data scientist would be able to quickly find relevant datasets among numerous data sources, whether that be by searching for joinable tables or finding more data entries or attributes to augment an existing table.

One way to achieve those goals is to use schema matching, which is the task of finding matching pairs of columns between a source and target schema. In this way, schema matching captures relationships between elements of different schemas, which can aid in dataset discovery. Schema matchers can be used to find different types of table relationships, such as unionable or joinable tables.

There are several different varieties of schema matchers. For instance, schema matchers can return *one-to-one* or *ranked* matches. One-to-one matchers simply match exactly one column in the target schema to each column in the source schema, while ranked matchers return a ranked list of potential matches in the target schema for each column in the source schema. Another way to categorize matchers is to consider the data the matcher takes as input. *Schema-based* matchers consider only metadata information, such as attribute names, data types, and other contextual information. Meanwhile, *instance-based* matchers consider only the value information of each column, such as value distributions and syntactic similarity. *Hybrid* matchers use both schema and instance information.

With so many schema matching variants, a cohesive way to compare and evaluate different schema matching techniques is needed. Valentine [7] evaluates a spectrum of

matching techniques and provides an experiment suite to execute automated matching experiments. For evaluation, Valentine implements seminal schema matching techniques and provides evaluation datasets.

This paper introduces a novel meta-learned schema matching technique, which expands on the meta-learning framework implemented by Rotom [12]. Through the use of meta-learning, this paper also explores the use of data augmentation for training a schema matching model.

The rest of this paper is organized as follows. Section 2 introduces the meta-learned schema matching technique and compares it against the matching techniques evaluated by Valentine. Section 3 presents the matching experiments and details the results. Finally, Section 4 provides an overview of the findings and offers ideas for future work.

2 Comparison of Schema Matching Techniques

2.1 Meta-Learned Schema Matching

The proposed meta-learned schema matching technique extends the Rotom meta-learning framework [12]. The Rotom framework uses meta-learning to learn a data augmentation policy that effectively fine-tunes a pre-trained language model, such as BERT [2], for some task. Previously implemented tasks include entity matching and error detection, as well as text classification tasks.

As with the previously implemented tasks, in order for schema matching to be compatible with the Rotom framework, the task is first translated into a sequence classification task. The input to the schema matching task is a pair of column values, where the output is a binary label in $\{0, 1\}$, with 1 indicating the columns are a match and 0 otherwise. The input pairs are serialized, and additional training samples are produced by applying data augmentation operations. Finally, all of the training samples are passed through meta-learning policy models to train the base matching model.

2.1.1 Serialization

To prepare the column pairs as input for the pre-trained language model, each column must first be serialized in a common form. Formally, suppose that each column c is represented by a series of values $\{a_1, a_2, ...a_n\}$. Then, c is serialized as follows: `[COL]` a_1 a_2 ... a_n

To serialize a pair of columns for input, each column is serialized separately and concatenated with the special token [SEP]. For a pair of columns (c, c'), serialization takes the form: `[COL]` a_1 a_2 ... a_n `[SEP] [COL]` a'_1 a'_2 ... a'_n

The following is an example of a serialized column pair from the test data:

```
[COL] Stockport MBC Stockport Stockport Stockport MBC Stockport Stockport
MBC Stockport MBC MBC Stockport MBC MBC [SEP] [COL] Human Soft & Arts
Fuel Soft Communication Equipment Vehicle Care Supplies Furnishings
Resources Furniture Community
```

Original Sequence	Augmentation #1	Augmentation #2
[COL] Appropriations Higher Reform rights; and Management Beverage on Special Serving Industry for on Funds Sugar	[COL] Appropriations Higher Reform rights; and Management Beverage Beer to Special Serving Industry for on Funds Sugar	[COL] Appropriations Higher Reform rights; Management On and Beverage Special Serving Industry for on Funds Sugar

Table 1: Examples of data augmentation on a test sequence.

Operator	Explanation
span_del	Delete a randomly sampled span of tokens
span_shuffle	Randomly sample a span and shuffle the tokens' order
attr_del	Delete a randomly chosen attribute and its value
attr_shuffle	Randomly shuffle the orders of all attributes
entry_swap	Swap the order of the two data entries

Table 2: Data augmentation operators proposed in Ditto [8].

2.1.2 Data Augmentation

After the data is processed and serialized, the resulting training set is expanded via data augmentation. Table 1 shows some examples of data augmentation on a serialized test column.

Data augmentation (DA) is the process of generating and labeling additional training samples from an existing training set. These new samples are produced by applying one or more *data augmentation operators*, which transform an original sample while preserving the original label. Successful data augmentation enlarges the training size while maintaining label quality and allows the model to learn invariance to some amount of noise. Additionally, multiple DA operators can be combined, chosen in accordance to a *data augmentation policy*.

There are a variety of simple DA operators that can be applied to text classification tasks. Ditto [8], which performs related data augmentation work, proposes a set of DA operators–listed in Table 2–for the task of entity matching.

In addition to these simple DA operators, Rotom [12] offers *InvDA*, a new DA operator based on a trained seq2seq model. The InvDA model is trained for the task of *inverting* some DA operator. Then, when given a non-augmented sample as input, the InvDA output becomes an augmented sample.

2.1.3 Meta-Learning Framework

After data augmentation, the full training set is ready to be used to train the schema matching classification model. However, there is no guarantee that all of the augmented samples are high-quality with un-corrupted classification labels. To address this, the meta-learning algorithm implemented by Rotom uses two policy models to learn a data augmentation policy [12].

The first policy model is the *filtering model*. The filtering model is a binary classifier that decides which augmented samples to keep. The classifier takes as input the original

class label and a measurement of difference between the target model's predicted label distribution on the original sequence and the augmented sequence. The intuition behind using these features is that too many changes likely indicates a semantics drift away from the original sequence. The augmented samples that make it through the filtering model are then passed on to the second policy model.

The second policy model is the *weighting model*. The weighting model's goal is to weight each example so that the target model can be better optimized. As input, the weighting model takes in a pre-trained encoding of the augmented sequence. As output, each example is assigned a weight such that harder examples have a higher weight.

2.2 Other Schema Matching Methods

2.2.1 Schema-Based Matchers

Cupid [10]

In Cupid, schemas are translated into tree structures. Each node of the tree structure represents a schema element, using information such as attribute names, data types, and constraints. To compute similarity, two matching techniques are weighted to produce a final result. The first is *linguistic matching*, which compares name similarity for each pair of elements that belong to the same category. The second is *structural matching*, which compares context similarity based on the tree structure schema representation.

Similarity Flooding [11]

In similarity flooding, schemas are translated into directed graphs. The nodes of the graph are schema elements, and the edges are the relationships between elements. To compare two schemas, the corresponding graphs are merged into a propagation graph. From there, similar nodes collapse into map pairs, which iteratively propagates similarity to the pair's neighboring nodes. The final output is a mapping between corresponding nodes of the input graphs.

COMA [3, 4]

COMA is a schema matching system that incorporates multiple schema-based matchers, which can be configured using provided combination strategies. The extension, COMA++ [4], augments COMA with two instance-based matchers.

2.2.2 Instance-Based Matchers

Distribution-based Matching [13]

The distribution-based matcher translates databases into graphs. The nodes of the graph are columns, and the edges are the column relationships, as determined by statistical measures. The algorithm can then cluster relational attributes using column value similarity. The final output is a set of disjoint clusters of related attributes.

Jaccard-Levenshtein Matcher

The Jaccard-Levenshtein matcher is a simple instance-based baseline. This matcher computes the Jaccard similarity for all column pairs, treating two values as identical if

their Levenshtein distance is below a given threshold. The final output is a ranked list of column pairs using the similarity scores.

2.2.3 Hybrid Matchers

EmbDI [1]

EmbDI is a method for embedding values and attribute names of relations. These embeddings are obtained by first translating the pair of datasets into a graph-based representation. Then, the embeddings are learned using sentences derived from these graphs in a way that preserves similarity between elements. For some challenging cases, external knowledge, such as synonym dictionaries, are used as well.

SemProp [5]

SemProp uses a two-stage schema matching approach using pre-trained word embeddings and a domain-specific ontology. The first stage is a *semantic matcher* that uses word embeddings to follow links between attribute and table names and their ontology classes. Element pairs that fail the semantic matcher are forwarded to a *syntactic matcher*.

3 Experiments

3.1 Methodology

The experiments in this project compare the effectiveness of the Rotom meta-learning schema matching techniques against existing instance-based matching techniques. To do this, four variants of Rotom-based models were trained on the same training data. These four experimental models are as follows:

- Rotom (full meta-learning framework) with semi-supervised learning
- Rotom (full meta-learning framework) without semi-supervised learning
- InvDA data augmentation and fined-tuned RoBERTa
- Fine-tuned RoBERTa

After training, each experimental model was evaluated and compared to the two instance-based matchers implemented by Valentine [7]—the distribution-based matcher [13] and the Jaccard-Levenshtein matcher.

3.2 Implementation and Baselines

For the Rotom-based models, the base language model used for matching is RoBERTa [9]. RoBERTa was chosen due to its effectiveness on the task of entity matching [8, 12].

The distribution-based matcher [13] and the Jaccard-Levenshtein matcher were chosen from Valentine as two additional baselines. These matchers were chosen because they have been well evaluated by Valentine [7]. Additionally, the Rotom models are instance-based matchers, making the two instance-based matchers from Valentine the most meaningful to compare against.

Dataset	Rotom w/ SSL	Rotom w/o SSL	InvDA + RoBERTa	RoBERTa	Distribution-Based	Jaccard-Levenshtein
SANTOS	0.7071	0.6757	0.7441	0.7445	0.4899	0.7104
Open Data	0.6060	0.8500	0.7310	0.6130	0.2963	0.8431

Table 3: Average F1 scores for each experimental and baseline model on the two test sets.

3.3 Data

3.3.1 Data Sources

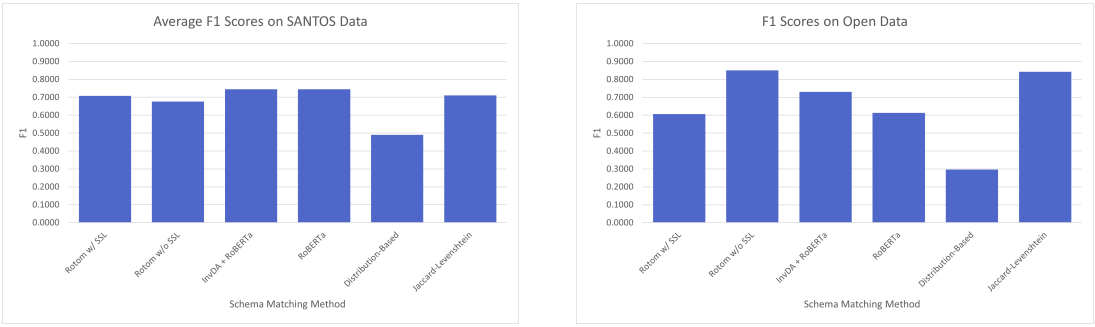
The primary source of data comes from the SANTOS small baseline [6], which contains data lake and query tables used for evaluating table union search. The data lake dataset contains 550 tables, and this dataset is adapted for the task of schema matching by sampling and serializing column pairs from these tables. The labeled data produced is split into test, train, and validation sets: 533 test samples, 2846 training samples, and 507 validation samples. The ground truth is determined by considering two columns to be a match if they have matching column names.

Another source of data comes from Valentine's Open Data table pair [7]. Valentine's evaluation uses two categories of datasets—fabricated and real-world—which are also published on the Valentine project website. The real-world datasets use real-world table pairs with a manually generated ground truth. On the other hand, the fabricated datasets use table pairs created from a single existing table with the original table as the ground truth. This fabrication process is automated using Valentine's fabrication module, which splits existing tables horizontally or vertically with a given amount of overlap. The fabrication module also offers the option of adding noise into the schema information and instance values.

Valentine applies this fabrication module with varying parameters on an Open Data table, among other table sources. The Open Data table chosen originates as the second table from the `base.sqlite` collection. The table pair used in this project was generated with a horizontal split with 50% row overlap and no noise added to the schema or instances. Column pairs were selected between the two tables to create an additional test set with 45 samples, with the ground truth determined by matching column names.

3.3.2 Data Pre-Processing

Each labeled column pair is pre-processed before serialization. First, any column pairs containing a numeric column are discarded. This is done because the Rotom-based models use a language model to perform the matching. Second, null values are dropped. Then, a token count of 15 is set for each column. If either column in the pair has fewer than 15 tokens, the pair is discarded. Otherwise, 15 random tokens are chosen from each column. Finally, the column pair is ready for serialization, as detailed in Section 2.1.1.



(a) Results on the SANTOS test data

(b) Results on the Open Data test data

Figure 1: Plots of average F1 scores on the two test sets.

3.4 Overall Results

The main results of the matching experiments are detailed in Table 3 and in Figure 1.

The matching experiments found that, on average, the Rotom-based models performed similarly to the Jaccard-Levenshtein baseline on the SANTOS data, where the RoBERTa baseline model surprisingly had the highest effectiveness with an average F1 score of 0.7445. Over the SANTOS data, the Rotom meta-learning framework decreases effectiveness compared to the RoBERTa baseline, while data augmentation via InvDA also does not improve much on the RoBERTa baseline.

In contrast, over the Open Data data, the most effective matcher was Rotom without semi-supervised learning, closely followed by the Jaccard-Levenshtein baseline. While the meta-learned model with semi-supervised learning was the least effective of the Rotom-based models for the Open Data data, the effectiveness of the other Rotom-based models followed the expected ranking. The meta-learned model without semi-supervised learning performed best, followed by data augmentation, followed by the baseline language model.

It is also interesting to note that the distribution-based model [13] was the least effective matching technique on both test sets.

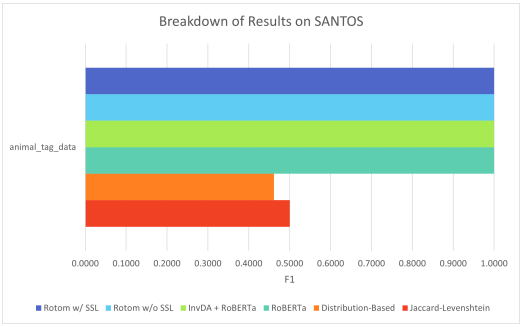
3.5 Results Breakdown and Discussion

The SANTOS data lake tables contain a diverse range of table subjects, and the effectiveness of each matcher varies greatly from table set to table set. For the full table of results, refer to the figures in Appendix A. This section will look in more detail at specific scenarios in the test data to discuss and compare the different matching techniques evaluated.

3.5.1 Scenario: Rotom-based models are more effective than baseline models

As seen in Figure 2, there is a noticeable difference in the performance between the Rotom-based models and the Valentine models on the `animal_tag_data` table set. The Rotom-based models all obtained an F1 score of 1. However, the Valentine models only obtained F1 scores between 0.4 and 0.5.

The breed-related columns are especially well-suited to the language model and data augmentation techniques. There is a fairly large domain of dog breeds, and the smaller set



(a) F1 scores

animal_ id	animal_ type	breed_ group	primary_ breed	tag_ no	tag_ type	tag_ subtype	tag_ stat
A577425	DOG	SETTER/ RETRIEVE	GOLDEN RETR	L16-044079	LIC ALTERED	HLP WEB	RENEWED
A583973	CAT	SHORTHAIR	DOMESTIC SH	L16-056016	LIC ALTERED	HLP SCAN	RENEWED
A562321	DOG	TOY	SHIH TZU	U15-003015	RABIES CERT	HLP IMPORT	CURRENT

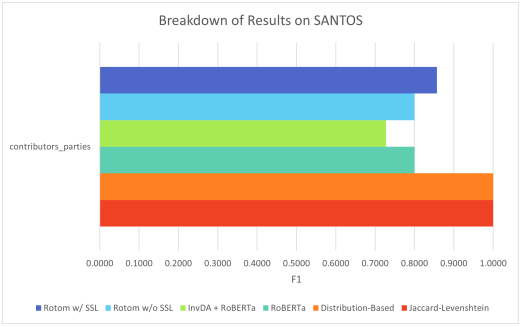
(b) Sample of a `animal_tag_data` table

Figure 2: Average F1 scores for each schema matching technique on the `animal_tag_data` table set.

overlap meant that the Jaccard-Levenshtein matcher was less effective. The same issue can be seen for the `animal_id` and `tag_no` columns. While the ids have a consistent format, there are a lot of character differences from value to value, and the Jaccard-Levenshtein matcher uses Levenshtein distance as a threshold to decide value equality.

3.5.2 Scenario: Baseline models are more effective than Rotom-based models

On the other hand, as seen in Figure 3, there is the opposite difference in performance on the `contributors_parties` table set. The two Valentine baselines obtained F1 scores of 1, while the Rotom-based models were less effective.

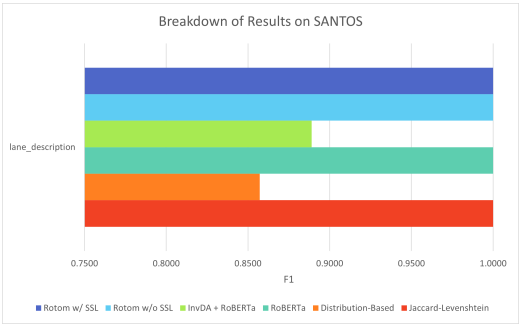


(a) F1 scores

Political Entity	Recipient	Electoral event	Fiscal date
Registered Party	Canadian Reform Conservative Alliance	Annual	12/31/2002
Registered Party	Canadian Reform Conservative Alliance	Annual	12/31/2003
Registered Party	New Democratic Party	Annual	12/31/2003

(b) Sample of a `contributors_parties` table

Figure 3: Average F1 scores for each schema matching technique on the `contributors_parties` table set.



(a) F1 scores

Sdate	Lane Description	Direction Description	Flag Text
19/07/2012 00:00	Northbound	NorthEast	Bad data
19/07/2012 00:00	Southbound	SouthWest	Bad data
19/07/2012 01:00	Northbound	NorthEast	Bad data

(b) Sample of a `lane_description` table

Figure 4: Average F1 scores for each schema matching technique on the `lane_description` table set.

In this table set, every column has a very small domain of values. Additionally, there is no overlap between domains among the columns in a table. This means that the Valentine baselines are both very effective on this data.

For the Rotom-based models, the small domain means that data augmentation can make the task harder by introducing noise. Moreover, the random sampling of column tokens during serialization may also increase the difficulty of the task. Specifically, between the `Political Entity` and `Recipient` columns, random sampling may result in oversampling the token *party*, which may throw off the classification.

3.5.3 Scenario: Rotom-based models and baseline models are both effective

In some cases, both the Rotom-based models and the baseline Valentine models had comparable effectiveness. Figures 4 and 5 look at two table sets that fit this scenario.

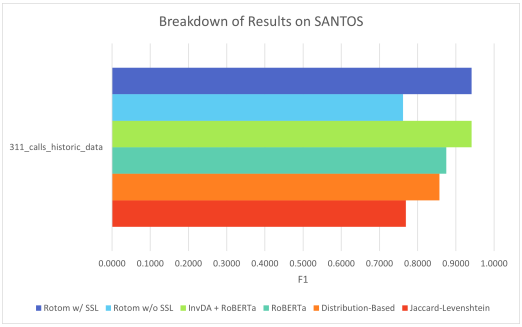
The first table set comes from the `lane_description` set of data lake tables. As seen in Figure 4, the two meta-learned matchers, the RoBERTa baseline, and the Jaccard-Levenshtein baseline all resulted in an F1 score of 1.

It is interesting to note that data augmentation greatly decreased the effectiveness of the RoBERTa matcher. This could be due to very small domains for the non-date columns, meaning that data augmentation may cause extra noise. However, the meta-learning framework was successfully able to overcome the negative impact of the data augmentation.

Looking at the Valentine models, the Jaccard-Levenshtein matcher was highly effective because the small domain is especially suited to the use of Jaccard similarity. On the other hand, the distribution-based matcher was the least effective matching technique on this table set. This was an unexpected result, since the column values are consistently formatted, in addition to having a small domain.

The second table set comes from the `311_calls_historic_data` set of data lake tables. As seen in Figure 5, while no matcher obtained an F1 score of 1, all of the matchers performed more consistently while achieving relatively good results.

The least effective matching technique on this table set is the Jaccard-Levenshtein matcher. This is likely due to most of the columns having a larger domain of values.



(a) F1 scores

issue_type	ticket_created_date_time	ticket_closed_date_time	ticket_status	neighborhood_district	city	state	case_title
Pothole/Roadway Surface Repair	7/28/2015 8:23:00 AM	7/29/2015 12:47:38 PM	Closed	MID-CITY	NEW ORLEANS	LA	Roadway Surface Repair - Pothole
Trash/Garbage Pickup	7/23/2012 10:22:46 AM	11/30/2012 5:15:08 PM	Closed	LAKEVIEW	NEW ORLEANS	LA	Start Trash Service
Trash/Garbage Pickup	10/23/2017 11:01:04 AM	10/25/2017 1:41:54 PM	Closed	LAKEWOOD	NEW ORLEANS	LA	Start Trash Service

(b) Sample of a `311_calls_historic_data` table

Figure 5: Average F1 scores for each schema matching technique on the `311_calls_historic_data` table set.

The distribution-based matcher performed similarly to the baseline RoBERTa model.

On this table set, data augmentation was the most effective matching technique, with semi-supervised meta-learning obtaining the same F1 score. One interesting note is the difference between the two meta-learning techniques. While the semi-supervised meta-learning framework could leverage the positive impact of the data augmentation, the original meta-learning framework was actually much less effective than the RoBERTa baseline. More exploration into the advantages of semi-supervised learning would be interesting, especially noting that neither of the two meta-learning models consistently outperforms the other.

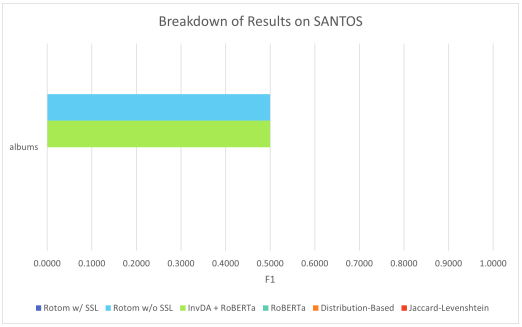
3.5.4 Scenario: Rotom-based models and baseline models are both ineffective

Finally, there are some tables for which schema matching is hard for all models evaluated. Figures 6 and 7 look at two table sets that fit this scenario.

The first table set comes from the `albums` set of data lake tables. On this table set, data augmentation and meta-learning obtain F1 scores of 0.5, while the rest of the matchers obtained F1 scores of 0.

As can be seen in Figure 6(b), both columns have a very large domain of values, with neither column having any consistent structure. Additionally, the two columns have overlapping values for some entities, as can be seen with "*stand up (for it)*" and "*stand up*". This makes the matching task very difficult for the Jaccard-Levenshtein and the distribution-based matchers.

This is another scenario where the two meta-learning frameworks have a major difference in results. The use of data augmentation had a positive impact to the RoBERTa baseline, and the original meta-learning framework was able to maintain that positive impact. However, the semi-supervised meta-learning framework was unsuccessful on this task.

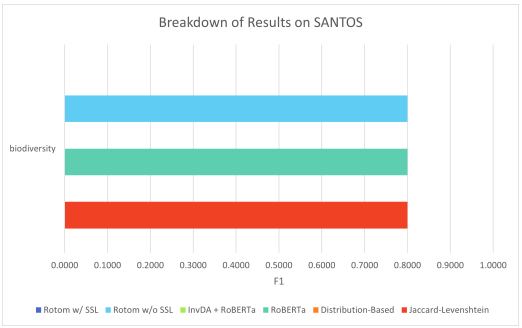


(a) F1 scores

title	album
mechanical ape!	charge!!
end credits (jfk homage)	songs in the key of springfield
stand up (for it)	stand up

(b) Sample of a `albums` table

Figure 6: Average F1 scores for each schema matching technique on the `albums` table set.



(a) F1 scores

scientific name	family name	common name
symphyotrichum novae-angliae	asteraceae	new england aster
phoradendron leucarpum	viscaceae	oak mistletoe
acer macrophyllum	aceraceae	bigleaf maple

(b) Sample of a `biodiversity` table

Figure 7: Average F1 scores for each schema matching technique on the `biodiversity` table set.

The second table set comes from the `biodiversity` set of data lake tables. This table set produced greatly varying results. The meta-learned matcher (without semi-supervised learning), the RoBERTa baseline, and the Jaccard-Levenshtein matcher all obtained F1 scores of 0.8, while the other three matchers obtained F1 scores of 0.

The meta-learning framework was able to match the effectiveness of the RoBERTa baseline, meaning that it successfully overcame the negative data augmentation influence. It is likely that the scientific names made the data augmentation less effective. Meanwhile, the Jaccard-Levenshtein matcher may have been able to perform fairly well because of overlapping parts of the scientific names.

4 Conclusion

This project introduces and evaluates the use of a meta-learning framework for schema matching. Building off of the Rotom meta-learning framework, four experimental language-model-based matchers are trained on the task of classifying column pairs as matches or non-matches. These experimental matchers are evaluated alongside the two instance-based matchers implemented by the Valentine experiment suite [7].

The results of the matching experiments found that no matching method performs consistently, and no matching method consistently outperforms the others. Instead, results vary greatly from dataset to dataset. These findings align with the results from the Valentine evaluation as well.

The matching experiments found that, on average, data augmentation with the InvDA operator does not improve effectiveness compared to the RoBERTa baseline. Additionally, on average, the Rotom meta-learning framework actually decreases effectiveness compared to the RoBERTa baseline. These results are surprising, since they indicate that the meta-learning framework is not producing a data augmentation policy that optimizes the target model.

For future work, it would be interesting to explore the effectiveness of data augmentation on other schema matching techniques. This could include expanding the column pair serialization to include schema information, or using whole table embeddings. One of the hybrid matchers from the Valentine paper, EmbDI [1] also uses column embeddings to determine matches. Another path for future work could focus on improving the data augmentation process for the current RoBERTa baseline. As can be seen in the inconsistent results between the two meta-learned schema matching techniques, there is more to explore with the Rotom framework. Additionally, future efforts may choose to experiment with the parameters, including the pre-processing phase. As one example, the token sampling process during serialization can be be improved.

While the current results on the effectiveness of a meta-learning framework for schema matching vary widely, the Rotom-based matchers are able to perform comparably with the Valentine baselines. This indicates some promise in applying meta-learning to other schema matching models.

References

[1] Cappuzzo, R., Papotti, P., and Thirumuruganathan, S. Creating embeddings of heterogeneous relational datasets for data integration tasks. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (2020), pp. 1335–1349.

[2] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[3] Do, H.-H., and Rahm, E. Coma—a system for flexible combination of schema matching approaches. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases* (2002), Elsevier, pp. 610–621.

[4] Engmann, D., and Massmann, S. Instance matching with coma++. In *BTW workshops* (2007), vol. 7, pp. 28–37.

[5] Fernandez, R. C., Mansour, E., Qahtan, A. A., Elmagarmid, A., Ilyas, I., Madden, S., Ouzzani, M., Stonebraker, M., and Tang, N. Seeping semantics: Linking datasets using word embeddings for data discovery. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)* (2018), IEEE, pp. 989–1000.

[6] Khatiwada, A., Fan, G., Shraga, R., Chen, Z., Gatterbauer, W., Miller, R. J., and Riedewald, M. Santos: Relationship-based semantic table union search, 2022.

[7] Koutras, C., Siachamis, G., Ionescu, A., Psarakis, K., Brons, J., Fragkoulis, M., Lofi, C., Bonifati, A., and Katsifodimos, A. Valentine: Evaluating matching techniques for dataset discovery. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)* (2021), IEEE, pp. 468–479.

[8] Li, Y., Li, J., Suhara, Y., Doan, A., and Tan, W.-C. Deep entity matching with pre-trained language models. *Proceedings of the VLDB Endowment 14*, 1 (sep 2020), 50–60.

[9] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. Roberta: A robustly optimized BERT pretraining approach. *CoRR abs/1907.11692* (2019).

[10] Madhavan, J., Bernstein, P. A., and Rahm, E. Generic schema matching with cupid. In *vldb* (2001), vol. 1, pp. 49–58.

[11] Melnik, S., Garcia-Molina, H., and Rahm, E. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings 18th international conference on data engineering* (2002), IEEE, pp. 117–128.

[12] Miao, Z., Li, Y., and Wang, X. Rotom: A meta-learned data augmentation framework for entity matching, data cleaning, text classification, and beyond. In *Proceedings of the 2021 International Conference on Management of Data* (2021), pp. 1303–1316.

[13] Zhang, M., Hadjieleftheriou, M., Ooi, B. C., Procopiuc, C. M., and Srivastava, D. Automatic discovery of attributes in relational databases. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data* (2011), pp. 109–120.

A Breakdown of All Results

Dataset	Rotom w/ SSL	Rotom w/o SSL	InvDA + RoBERTa	RoBERTa	Distribution-Based	Jaccard-Levenshtein
311_calls_historic_data	0.9412	0.7619	0.9412	0.8750	0.8571	0.7692
abandoned_wells	1.0000	0.7692	1.0000	1.0000	0.6667	0.8000
albums	0.0000	0.5000	0.5000	0.0000	0.0000	0.0000
animal_tag_data	1.0000	1.0000	1.0000	1.0000	0.4615	0.5000
biodiversity	0.0000	0.8000	0.0000	0.8000	0.0000	0.8000
business_rates	0.8485	0.6122	0.9143	0.6923		
cdc_nutrition_physical_activity	0.6000	0.7368	0.8000	0.8333	0.6000	0.7273
cihr_co-applicant	0.8889	0.5714	0.7500	0.8571	0.4615	0.8235
civic_building_locations	0.5000	0.6154	0.6667	0.8571		
complaint_by_practice	1.0000	0.5455	0.8000	0.4615	0.0000	0.8571
contributors_parties	0.8571	0.8000	0.7273	0.8000	1.0000	1.0000
data_mill	0.7500	0.6667	0.8750	0.6667		
deaths_2012_2018	0.8421	0.8333	0.8696	0.7407		
film_locations_in_san_francisco	0.0000	0.0000	0.5000	0.8333	0.0000	0.2857
immigration_records	0.8000	0.5000	0.7143	0.8571	0.6667	0.6667
ipopayments	0.4000	0.7000	0.8000	0.8889	0.5455	0.6667
job_pay_scales	0.7273	0.6154	0.7692	0.9231	0.4444	0.9231
lane_description	1.0000	1.0000	0.8889	1.0000	0.8571	1.0000
mines	0.8571	0.8571	1.0000	0.7500	0.8571	0.6667
monthly_data_feed	0.5000	0.6667	0.4444	0.4444	0.0000	0.0000
new_york_city_restaurant_inspec	0.8421	0.5714	0.6400	0.6897	0.4615	0.7500
oil_and_gas_summary_production	0.8889	0.9091	0.8000	0.6667	0.5714	0.7500
practice_reference	0.8000	0.6667	0.7692	0.7059	0.5000	1.0000
prescribing	0.5000	0.8000	0.6667	0.6667	0.5000	0.9091
psyckes_antipsychotic_polypharm	0.7500	0.8000	0.8333	0.7143	0.5714	1.0000
purchasing_card	0.8750	0.7619	0.8421	0.6400	0.5000	0.7143
report_card_discipline_for_2015	0.8000	0.7500	0.8000	0.7500	0.5714	0.6154
senior_officials_expenses	0.2000	0.5926	0.5926	0.5600	0.0000	0.0000
stockport_contracts	0.6429	0.6829	0.6400	0.6275	0.2727	0.9444

time_spent _watching _vcr_movies	1.0000	0.5000	0.8571	0.7500	1.0000	1.0000
tuition_assistance _program_tap	0.8889	0.5455	0.7143	0.7143	0.7500	0.8889
wholesale _markets	0.7273	0.8235	0.7143	0.7368	0.6000	0.8333
workforce _management _information	0.8451	0.3529	0.8451	0.9873		
ydn_spending_data	0.7692	0.6667	0.6250	0.8235		
SANTOS Average	0.7071	0.6757	0.7441	0.7445	0.4899	0.7104
Open Data	0.6060	0.8500	0.7310	0.6130	0.2963	0.8431

Table 4: This table shows the F1 scores by schema matching technique for each subset of tables used as test data in the matching experiments. The SANTOS table sets are listed first, followed by the overall SANTOS average. The Open Data test data results are also listed at the end.

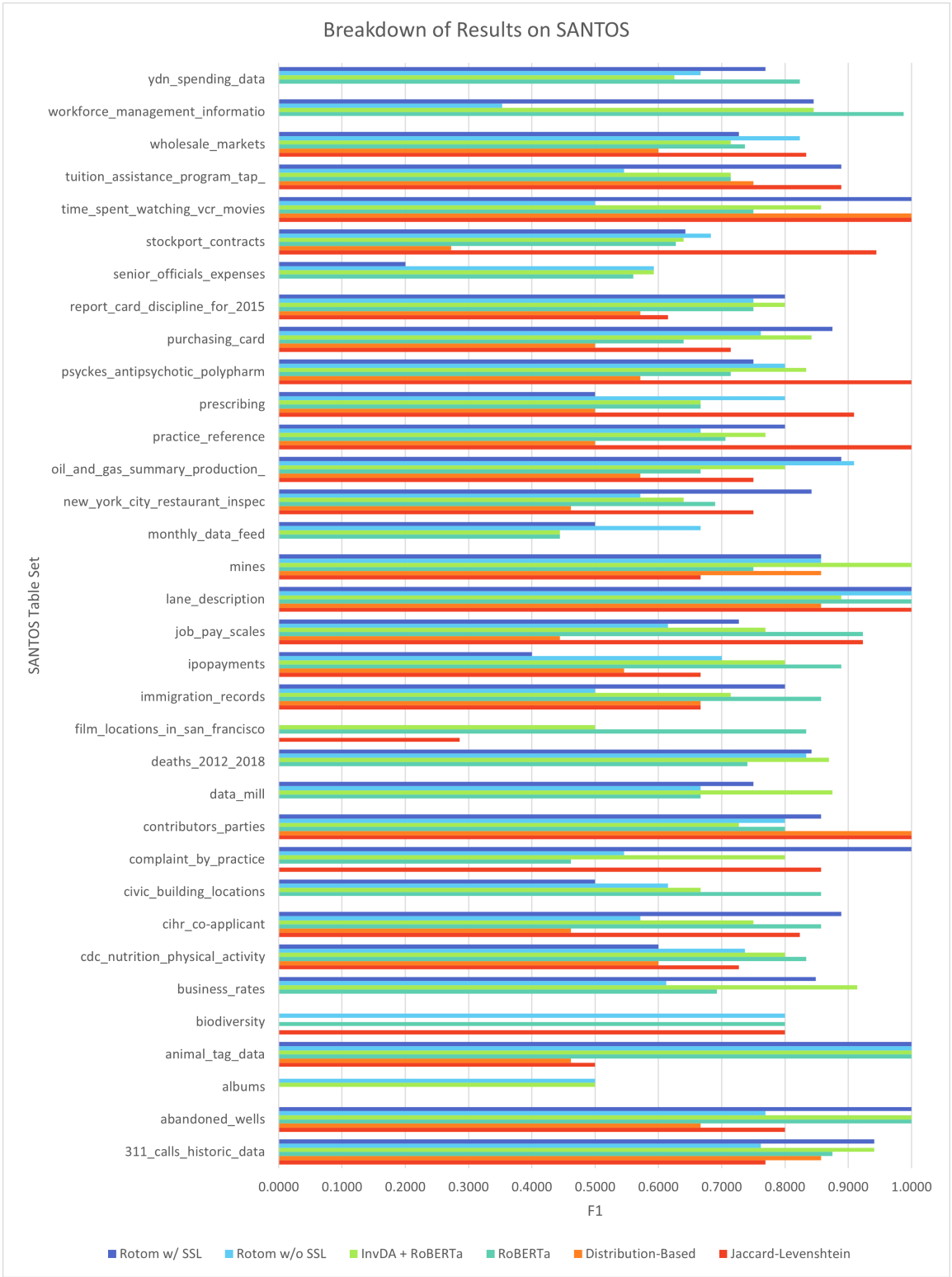


Figure 8: This chart compares the F1 scores of each schema matching technique, broken down by test table set.