Rotom: A Meta-Learned Data Augmentation Framework for Entity Matching, Data Cleaning, Text Classification, and Beyond

Zhengjie Miao*
Duke University
Durham, NC, USA
zjmiao@cs.duke.edu

Yuliang Li
Megagon Labs
Mountain View, CA, USA
yuliang@megagon.ai

Xiaolan Wang
Megagon Labs
Mountain View, CA, USA
xiaolan@megagon.ai

ABSTRACT

Deep Learning revolutionizes almost all fields of computer science including data management. However, the demand for high-quality training data is slowing down deep neural nets' wider adoption. To this end, data augmentation (DA), which generates more labeled examples from existing ones, becomes a common technique. Meanwhile, the risk of creating noisy examples and the large space of hyper-parameters make DA less attractive in practice. We introduce Rotom, a multi-purpose data augmentation framework for a range of data management and mining tasks including entity matching, data cleaning, and text classification. Rotom features InvDA, a new DA operator that generates natural yet diverse augmented examples by formulating DA as a seq2seq task. The key technical novelty of Rotom is a meta-learning framework that automatically learns a policy for combining examples from different DA operators, whereby combinatorially reduces the hyper-parameters space. Our experimental results show that Rotom effectively improves a model's performance by combining multiple DA operators, even when applying them individually does not yield performance improvement. With this strength, Rotom outperforms the state-of-the-art entity matching and data cleaning systems in the low-resource settings as well as two recently proposed DA techniques for text classification.

ACM Reference Format:
Zhengjie Miao, Yuliang Li, and Xiaolan Wang. 2021. 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 (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3448016.3457258

1 INTRODUCTION

The burst of deep learning systems has led to significant advancement in many fields such as natural language processing (NLP), computer vision (CV), robotics, and more [19, 73, 93]. The database community is no exception [42, 61, 85, 92]. For example, in data integration, one can achieve the state-of-the-art (SOTA) accuracy of entity matching (EM) [49] by casting it as a sequence-pair classification task and training a deep neural net on a downstream EM dataset. While deep learning has been proven effective, the high demand for both the quality and quantity of training data is slowing down deep learning systems' wider adoption. To this end, the recent success of pre-trained models [19, 41, 53] has partly addressed this issue by allowing the developer to first *pre-train* a model on a large unlabeled data corpus then *fine-tune* the model on a relatively smaller labeled dataset of the downstream task. However, the problem is only partially solved as the fine-tuning approach still requires a non-trivial amount of high-quality labeled examples (e.g., ~10k for a typical EM application [49, 61]). Given the expensive cost of obtaining quality labeled data, there has been significant interest in effortless data collection [47, 69].

Motivated by the above considerations, in this paper, we study the problem of training high-quality deep learning models while *requiring only a small number (e.g., 200) of labeled examples*. We develop a multi-purpose system, Rotom, that supports a range of data management and mining tasks including entity matching, data cleaning, text classification. Rotom achieves this versatility by having a simple architecture of fine-tuning LMs, thus it can support any tasks that can be formulated as sequence classification.

Rotom reduces its requirement of labeled data via Data Augmentation (DA). Widely used in CV and NLP, DA automatically generates additional labeled training data from existing ones. In general, a DA operator transforms a training example into a different version while preserving the classification label (e.g., image rotation for CV tasks or replacing a word with its synonyms for NLP tasks), and thus it enlarges the training set and allows the model to learn different invariance properties. Multiple DA operators can be combined into a *DA policy* to achieve better results. Rotom exploits augmented data in a novel way following a meta-learning paradigm as it learns a policy for selecting and assigning weights to augmented examples to better train the target LM.

Challenges. Developing generally effective DA operators for a wide range of applications as in Rotom is not an easy task. Below we illustrate two major limiting factors of existing data augmentation techniques and give insights into how we address them in Rotom.

First, although data augmentation is shown to be effective by increasing the diversity of training data, it has the risk of changing the ground-truth labels. Training with these corrupted examples can potentially damage the quality of the target LM.

Example 1.1. Consider classifying the intent of a sentence:

Where is the Orange Bowl? [Intent: Location]

Even applying only one single DA operator can generate "wrong" examples that do not preserve the original label. For example, if

*The majority of this work was done during an internship at Megagon Labs

SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3457258

one simply replaces the word "where" with another interrogative pronoun "what", the resulting sentence will have a different meaning and the intent becomes asking for a description. Similarly, if one inserts the word "from" after "bowl", which is very likely to happen when using a language model to predict the insertion, the question intent will also become asking for a description. It will be more likely to get "wrong" examples if one applies a sequence of DA operators to get more diverse augmented examples.

Restricting DA operators to those with the least risk of changing the meaning of the sentence (e.g., replacing a word with its synonym from a dictionary) can avoid this issue, but will generate examples that are almost the same as the original ones and thus lead to less performance gain. Therefore, there is a trade-off between diversity and quality for data augmentation: for methods that generate more diverse examples like applying multiple operators at one time, the risk of label corruption is higher [86]. Tuning this trade-off is important to improve the performance of the model. Rotom addresses this challenge by (1) introducing a new DA operator, InvDA, which can generate diverse yet natural examples by formulating DA as a seq2seq problem; and (2) devising a filtering/weighting model that selects the good augmented examples from the pool of potentially corrupted augmentations.

Second, it is non-trivial to find the most effective DA operators or policies. DA operators often introduce a new set of hyper-parameters. For example, a simple word replacement transformation has at least two hyper-parameters: (1) the sampling method to choose the word to be replaced and (2) dictionary [59] or similarity-based method [60] for choosing the synonyms for replacement. In practice, the developer needs to keep trying combinations and re-training the model until the result is satisfactory. This process is inefficient and relies on heuristics, often resulting in a sub-optimal choice. There have been efforts on automatically searching for effective DA policies [16, 51] in CV. However, these search algorithms are usually fixed to a template [32, 44], e.g., two sequential simple transformations, thus does not support operators beyond simple transformations, nor prevents them from generating bad examples.

We address this challenge by considering a more general setting: instead of combining DA operators, Rotom combines *augmented examples* generated by multiple DA operators. The operators can be simple transformations, generation-based operators such as InvDA, or any complex learned operators. Without manual tuning of hyper-parameters, Rotom leverages meta-learning to automatically learn an optimized policy for (1) filtering out noisy augmentations and (2) re-weighting the remaining examples based on their importance. As a result, Rotom effectively selects and combines the high-quality portion from each operator to better train the target model.

Contributions. We make the following contributions in this paper.

- We present Rotom, a multi-purposed data augmentation system for a range of data management and mining tasks including entity matching, data cleaning, and text classification. Rotom models these tasks as sequence classification and solves them by fine-tuning pre-trained language models (LMs) such as BERT [19]. The architecture of Rotom is depicted in Figure 1.

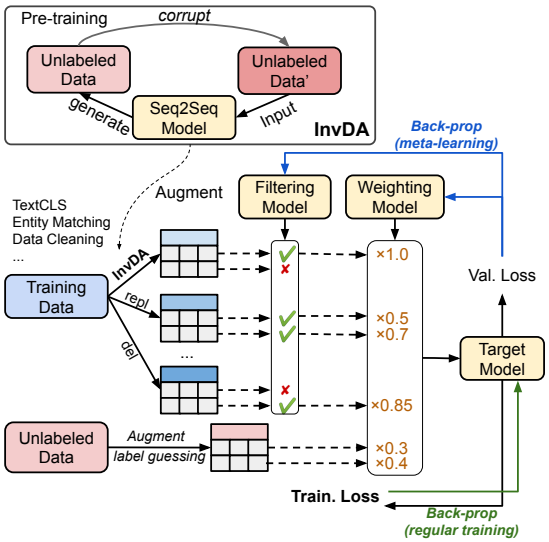

Figure 1: The Rotom DA framework for entity matching, error detection, text classification, and more. The InvDA operator of Rotom learns a seq2seq DA model via reconstructing corrupted unlabeled sequences. Rotom also features a semi-supervised meta-learning framework for filtering and weighting augmented and unlabeled examples. The filtering+weighting models are jointly trained with the target model by descending to a low validation loss.

- We introduce in Rotom a versatile data augmentation operator InvDA. Based on a seq2seq generative model, InvDA can generate natural and diverse augmented sequences arbitrarily different from the original sequence. Through a fully automated self-training process, we allow InvDA to learn how to augment by inverting the effect of multiple simple transformations.
- Rotom features a meta-learning framework that filters and weights a large number of augmented examples to assemble them into high-quality training signals. Instead of hand-crafting a DA policy, Rotom automatically optimizes the filtering/weighting model jointly with the target model. As a result, the filtering/weighting model "learns" how to better select and combine the augmented examples to improve the overall performance of the target model, thus addresses the diversity-quality trade-off in DA.
- We show that Rotom can be naturally extended to the *semi-supervised learning* (SSL) setting to exploit the large amount of unlabeled data. While the substantial quality variance of the guessed labels is a major challenge in traditional SSL, the meta-learning framework enables Rotom to select only the high-quality guesses, thus further boost the performance of the target model.
- We conducted extensive experiments to evaluate Rotom on benchmark datasets across all 3 supported tasks. Our experimental results show that Rotom effectively combines simple DA operators, InvDA, and unlabeled examples even when applying them independently does not yield a performance gain. With this capability, Rotom outperforms (1) the SOTA EM systems by up to 6% F1 score, (2) the SOTA error detection system by 7.64% F1, and (3) two recent low-resource text classification techniques.

Paper outline. The rest of the paper is organized as follows. Section 2 reviews the supported tasks and Section 3 presents the InvDA operator. We introduce our meta-learning framework in Section 4.

We extend this framework to the semi-supervised setting in Section 5. Section 6 presents the experimental results. Finally, we discuss related work in Section 7 and conclude in Section 8.

2 PRELIMINARIES

In this section, we first review and formally define the tasks considered in our paper. Next, we introduce a baseline method of fine-tuning pre-trained language models (LMs) for these tasks. We also summarize the basic data augmentation operators which generate additional training data for optimizing the fine-tuning baseline.

2.1 Problem definition: sequence classification for entity matching and data cleaning

Rotom targets tasks of a general form of multi-class sequence classification. Formally,

Definition 2.1 (Sequence classification). Let $V = \{c_1, \ldots, c_k\}$ be a vocabulary of class labels. A sequence classifier M takes as input a sequence of tokens $S = [t_1, \ldots, t_n]$ and outputs a label $M(S) \in V$ of the input sequence S.

The above definition is commonly used to formulate the *text classification* problem in NLP which has a wide range of applications. For example, in sentiment analysis, the input sequence can be a product review, and the classifier predicts the binary sentiment class (i.e., positive or negative) of the review. Table 1 summarizes a few examples of applications of text classification.

Table 1: Example text classification tasks in NLP.

NLP Task	Input	Example Vocabulary
Sentiment Analysis	Product review	{1-star, 2-star, …, 5-star}
Topic Modeling	News article	{Politics, Sports, Technology, …}
Intent Classification	NL query	{Purchase, Subscribe, Inquiry, …}

While the above setting is most commonly used in NLP, it also applies to many data management tasks. Next, we review two such tasks: *entity matching* and *data cleaning*.

Entity matching (EM). Given two collections of data entries, EM [12] seeks to identify pairs of data entries that refer to the same real-world entity. An EM workflow [37] has two main steps: blocking and matching. The blocking phase typically uses simple heuristics (e.g., a pair of matching records must share at least 1 token) to identify a relatively small set of candidate pairs. Next, the matching phase classifies whether each candidate pair is a real match or not. Formally, suppose that each data entry e is represented by a set of key-value pairs $\{(\text{attr}_i, \text{val}_i)\}_{1 \le i \le m}$. Given a pair of data entries (e, e'), a matching model M outputs a label in $\{0, 1\}$ where $M(e, e') = 1$ if (e, e') is a match and 0 otherwise.

The matching problem can be effectively solved by formulating it as a sequence classification task [49]. Following [49], we serialize and concatenate all attributes/values from both entries into a single sequence. For example, consider two entries {Name: "Google LLC", phone: "(866) 246-6453"} and {Name: "Alphabet inc", phone: "6502530000"}, we can serialize them as

"[COL] Name [VAL] Google LLC [COL] phone [VAL] (866) 246-6453 [SEP] [COL] Name [VAL] Alphabet inc [COL] phone [VAL] 6502530000"

then we can apply the sequence classification techniques to solve the matching task. Note that we insert special tokens [COL] and [VAL] to indicate the starts of an attribute or a value. The special token [SEP] separates the two entries.

Data cleaning. Given a collection of relational tables, the goal of data cleaning [29, 33, 54, 55] is to identify and repair any errors in the tables such as typos, data formatting errors, or constraint violations. We focus on the first part of the problem of identifying erroneous entries, which is also known as *error detection* [29, 55]. Table 2 shows an example instance of the error detection problem.

Table 2: Error Detection in an employer table (errors are in red).

Name	Address	phone
Google LLC	1600 amphitheatre pkwy	(877) 355-578
AlphaBet inc.	1600amphiteatrepkwy	6502530000
Apple Inc.	One Infinite Loop	(408) 606-5775

In Rotom, we cast error detection into sequence classification like for EM. We can serialize the cell value that we would like to check for correctness by concatenating with special tokens. For example, for the last cell of Table 2, we generate the sequence "[COL] phone [VAL] 6502530000". To detect *context-independent* errors, i.e., errors related only to the cell value, we can use this string as the input to the sequence classifier. For *context-dependent* errors, we serialize the entire row as the "context" and append to the end the cell of interest (separated by "[SEP]"). For example,

"[COL] Name [VAL] Apple Inc. [COL] Address [VAL] One Infinite Loop [COL] phone [VAL] (408)606-5775 [SEP] [COL] phone [VAL] (408)606-5775"

Discussion on serializing data entries. Because of its simple formulation, Rotom supports any tasks that can be cast into sequence classification. This makes Rotom a flexible solution. For example, as shown in [49], serializing data entries for EM allows the input entities to have arbitrary schema thus it does not require schema alignment before matching. However, serialization does not always yield the best performance. In error detection, for example, there can be error types that violate table-level constraints such as aggregate constraints and functional dependencies. Capturing such constraints requires serializing the whole table which can easily exceed the max sequence length allowed by the LM. In this sense, Rotom is better at tasks serializable to sequences that are mostly textual and of medium length (e.g., hundreds of tokens).

2.2 Baseline: fine-tuning language models

Fine-tuning pre-trained language models (LMs) has been shown a simple yet effective method for sequence classification [19, 67]. Pre-trained LMs such as BERT [19] and GPT-2/3 [8, 66] have demonstrated good performance on a wide range of NLP tasks. These models are typically neural nets consisting of Transformer layers [83], typically 12 or 24 layers, and pre-trained on large text corpora such as Wikipedia by self-supervised learning. During pre-training, the model is self-trained to perform auxiliary tasks such as missing token and next-sentence prediction which allows the model to learn to capture the lexical or semantic meanings of the input sequences.

One can leverage the pre-trained LM in a downstream task by fine-tuning it on a task-specific training set. For sequence classification, this can be done by the following steps:

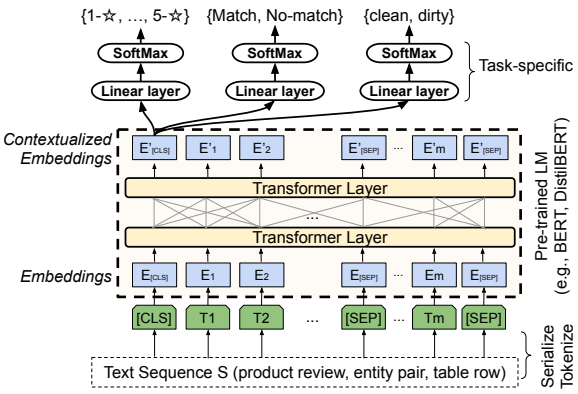

Figure 2: The model architecture of a pre-trained LM in Rotom**.**

(1) Add task-specific layers after the last layer of the LM. Specifically, we add a fully connected layer whose output dimension equals to the number of classes and a softmax classifier as the final layer to perform multi-class classification.
(2) Initialize the modified network N with parameters from the pre-trained LM.
(3) Train N on the training set until convergence.

Figure 2 shows the model architecture of an LM. The model consists of (1) a token embedding layer, (2) the Transformer layers which use the self-attention mechanism to capture sequence semantics [83], and (3) the added task-specific layers (linear followed by softmax) for the downstream classification task. Conceptually, the [CLS] token "summarizes" all the contextual information in the entire sequence needed for classification as a contextualized embedding vector $E'_{[\text{CLS}]}$ which can be fed to the task-specific layers as input features for classification [19].

2.3 Simple data augmentation operators

The success of the above fine-tuning approach heavily depends on the amount of high-quality labeled training data. When there is an insufficient amount of training data, *Data Augmentation* (DA) is a commonly used technique. For example, in computer vision, one can obtain additional training examples by performing simple transformations such as rotating or flipping on an original training image and expect that the label of the image stays unchanged. For text classification, such simple transformations can be insertion or deletion of a single token. In a nutshell, DA consists of a transformation operator that preserves the label of the original training example.

Besides enlarging the training set, DA improves the robustness of the trained models against data noise as the models are forced to "learn harder". For example, by applying the deletion operator, the model learns the invariance property that the meaning of a sentence stays unchanged if a single token is dropped.

Data augmentation has been shown effective for all the 3 supported tasks of Rotom [29, 49, 86]. We summarize in Table 3 the simple augmentation operators that we have implemented in Rotom. Each operator transforms the input sequence at a different level: token, span, column/attribute, or entity. The sampling of the tokens can be uniform sampling or sampling by the importance of each token. We measure the importance of a token by its inverse document frequency (IDF) so that less important tokens are more likely to be replaced/deleted. We obtain the synonyms of a token by wordnet [60]. We note that this list is by no means complete. Users of Rotom have the option of adding transformations customized for a specific task.

Table 3: Simple DA operators in Rotom**. Note that the attribute-level operators (col_shuffle and col_del) only apply to EM and Error-Detection. The entity_swap operator only applies to EM.**

Operators	Details
token_del	Sample and delete a token
token_repl	Sample a token and replace it with a synonym
token_swap	Sample two tokens and swap them
token_insert	Sample a token and insert to its right a synonym
span_del	Sample and delete a span of tokens
span_shuffle	Sample a span of tokens and shuffle their order
col_shuffle	Choose two columns/attributes and swap their order
col_del	Choose a column/attribute and drop it entirely
entity_swap	Swap the order of the two entity records

Limitations. Several factors are limiting the effectiveness of the above simple DA operators in practice.

- First, the examples generated by these DA operators are *unnatural*, i.e., deviate from the original sequence distribution. For text classification, the generated sequences may have grammar errors and thus may be undesirable for certain tasks.
- Second, the simple DA operators fail to generate more diverse examples while still preserving the class labels. For example, the sequences generated by the token_repl differ from the original sequence by at most 1 token. The span level operators might change the sequence by too much and thus corrupts the label.
- Third, combining these DA operators introduces a large space of hyper-parameters. The developer needs to choose from a pool of operators together with different operator-specific options. As a result, it became a common practice in previous work [49, 58, 86] to only enumerate and pick the best-performing single DA operator which can potentially lead to sub-optimal performance.

3 SEQ2SEQ DATA AUGMENTATION

Next, we introduce InvDA, a new DA operator for Rotom to generate natural yet diverse augmented sequences. InvDA is a seq2seq model trained on a task-specific sequence corpus.

3.1 Seq2seq models

A seq2seq (sequence-to-sequence) model [80] M_{gen} takes as input an *input sequence* s and generates a new sequence $t = M_{\text{gen}}(s)$ which we call the *target sequence*. Seq2seq models are commonly used in NLP tasks such as Machine Translation or Text Summarization whose output is an entire sequence instead of a class label (i.e., these tasks are generative instead of discriminative). The model typically consists of an encoder and a decoder, where the encoder takes s and maps it to a fixed-size vector representation, while the decoder predicts tokens sequentially based on this vector representation to generate a target sequence t.

Seq2seq models have been used for text data augmentation [44, 90]. The main advantage of formulating data augmentation as a sequence generation problem is the model's flexibility. Unlike simple transformations that limit the number of changes to the

original sequence, a seq2seq model has an unrestricted output space meaning that its output can be arbitrarily different from the input. However, there are still disadvantages of using these seq2seq models since they are not trained on the domain of the downstream tasks. For example, we cannot directly apply seq2seq models for text data [44, 90] to EM and EDT tasks, because the sequences in EM and EDT are usually not sentences and such seq2seq models trained on large natural language corpus may not generate sequences of a distribution close to EM and EDT inputs. Hence, there is a need for seq2seq-based DA methods trained on task-specific corpus to generate more diverse sequences.

3.2 Inverse data augmentation

Next, we describe our seq2seq-based DA method InvDA. As aforementioned, the InvDA model needs to be trained properly to ensure the quality of the augmented sequences. The training data of a seq2seq model consists of a collection of input-target sequence pairs. Training a high-quality seq2seq model typically requires a large collection of such pairs. For example, a typical EN-DE translation model [83] is trained on more than 4.5M pairs of English-German sentence pairs of correct translations. To train InvDA for a downstream task, ideally, one needs to provide input-target pairs where the input is an original sequence of the task and the target is a "good" augmentation. An augmented sequence is good if it is:

(1) semantics-preserving,
(2) coming from the same sequence distribution as the input, and
(3) different from the input sequence as much as possible.

In practice, however, such input-target pairs are hard to obtain automatically without human annotations. Therefore, we propose a novel self-supervision method to train the InvDA model.

The intuition behind our method is that although high-quality augmented examples are hard to obtain, the *original training examples* are potentially high-quality augmentation for *some input sequences*. Moreover, we can obtain those sequences by *corrupting* the original sequence which can be done by applying multiple DA operators (dropping tokens, swapping token orders, etc.). We formally describe this process in Algorithm 1.

Algorithm 1: Training inverse data augmentation

input : A training set S; a set D of DA operators;
The number of operators n to be applied;
The hyper-parameters H (optimizer, learning rate, etc.)
output : A trained seq2seq model M_{gen}
variables : The input-target pairs P for training M_{gen}

$P \leftarrow \emptyset$;
for *sequence* $s \in S$ **do**
  $(\text{input}, \text{target}) \leftarrow (s, s)$;
  /* Corrupt s by random augmentations */
  for $i = 0$ *to* $n - 1$ **do**
    $d \leftarrow \text{RandomSample}(D)$;
    $\text{input} \leftarrow d(\text{input})$;
  $P \leftarrow P \cup \{(\text{input}, \text{target})\}$;
$M_{\text{gen}} \leftarrow \text{Train}(H, P)$;
return M_{gen};

Conceptually, the seq2seq model M_{gen} learns how to *invert* or *restore* the effect of corrupting a sequence by multiple simple DA operators. At prediction time, we apply M_{gen} to transform an original input sequence s. The training examples of M_{gen} satisfy requirements (2) and (3) above by construction as the target sequences are from the original training set and can be arbitrarily different from the input sequences by controlling the number of applied augmentations. Requirement (1) of semantic preserving is less obvious. The intuition is that the corruption process typically "removes" information from the input sequence (e.g., removing "sandwich" from *"I want to eat a sandwich"* provides an input-target pair *"I want to eat a"*-*"I want to eat a sandwich"* for training). At prediction time, we can expect an inverted effect of adding information, e.g., given another sequence in the original training set *"I want to eat at the cafe"* as input, the seq2seq model can generate *"I want to eat sandwich at the cafe"* which likely preserves the meaning from the input and is closer to the distribution of the original training set than the output of simple DA operators. We observe this effect in our examples in Tables 4 and 5.

Choice of models and generation methods. While the above method allows fully automatic training of InvDA, it is still unfeasible to train the seq2seq model from scratch especially when the training set S is small. We again address this issue by leveraging pre-trained models. The recently proposed pre-trained seq2seq models such as BART [46], T5 [67], and GPT-3 [8] can be fine-tuned to downstream tasks with a relatively small amount of training data. We choose the 12-layer T5-base model [67] in our implementation because of its relatively small size and the generalizability to multiple tasks including machine translation, summarization, and question answering.

For other DA methods based on sequence generation, please see Section 7 for a discussion of these techniques and comparison with InvDA. We also compare experimentally InvDA and a previous generation-based method [44] for text classification in Section 6.5.

3.3 Examples

We illustrate examples of InvDA in Tables 4 and 5 for the 3 supported tasks of Rotom. For text classification, all the 3 sequences generated by InvDA are good augmentations as they preserve the semantic meaning of location seeking while being very different from the original sequence in a natural way. In the error detection example, InvDA (to our surprise) can generate natural fake movie names from a real movie name "The DUFF" so that they preserve the correctness of the original data entry. For EM, InvDA rewrites the term "relational databases" into meaningful terms of "databases", "database systems", and "open-source databases". On the other hand, we observe that sequences generated by simple transformations are somewhat unnatural or deviate from the original meaning. Although InvDA can generate more natural and diverse examples, the original label is not guaranteed to be preserved, thus further filtering and weighting the augmented examples are still necessary.

4 META-LEARNING TO SELECT AND COMBINE AUGMENTED EXAMPLES

The performance of a machine learning model largely depends on the data that the model is trained on. Thus, when we apply

Table 4: Example augmentations by simple DA (DA1,2) and InvDA (InvDA1-3). We highlighted the changes in green and red.

	Text Classification - question intent	Error Detection - cleaning movie data
original	Where is the Orange Bowl ?	[COL] Name [VAL] The DUFF
DA1	Where is the Orangish Bowl ?	[COL] Name [VAL] DUFF
DA2	is the Orange Bowl ?	[COL] Name [VAL] DUFF The
InvDA1	Where is the Indianapolis Bowl in New Orleans?	[COL] Name [VAL] The DUFF (The Wrestling Wizard)
InvDA2	Where is the Orange Bowl held every February?	[COL] Name [VAL] The DUFF: The Adventures of Lena Green
InvDA3	Where is the Syracuse University Orange Bowl?	[COL] Name [VAL] The Duff Boy With The Devil

Table 5: Examples of InvDA for EM. The "_" symbol indicates a deleted token from the original sequence.

	EM - DBLP-ACM paper matching
original	[COL] title [VAL] effective timestamping in relational databases
DA1	[COL] title [VAL] effective _ in relational databases
DA2	[COL] title [VAL] effective relational in databases timestamping
InvDA1	[COL] title [VAL] effective timestamping in databases
InvDA2	[COL] title [VAL] effective timestamping in database systems
InvDA3	[COL] title [VAL] effective timestamping in open-source databases

DA (simple transformations or InvDA), it is critically important to ensure that the augmented examples are of high-quality. In Rotom, we achieve this goal by formulating it as an optimization problem. We set the optimization goal of *finding a set of augmented examples generated by any DA operators* that optimizes the performance of the target model trained on this set. Note that this is a more general setting than selecting a subset of DA operators to apply. For a DA operator that generates diverse and noisy examples, we expect Rotom to separate the "good portion" from noisy examples instead of taking or leaving all examples generated by the operator entirely.

Formally, for a language model M and a set S of training data, we denote by $\mathsf{Train}(M, S)$ the model M fine-tuned on S:

Problem 1 (DA optimization). *Given a (text classification, entity matching, error detection) model M to be trained, a labeled training set S_{train}, a validation set S_{val}, a set of DA operators D, and a function* loss *to minimize, find a subset $S_{\mathsf{aug}} \subseteq \bigcup_{d \in D}\{d(s)|s \in S_{\mathsf{train}}\}$ that minimizes* $\mathsf{loss}(\mathsf{Train}(M, S_{\mathsf{aug}}), S_{\mathsf{val}})$ *where* $\mathsf{loss}(M, S)$ *computes the loss value of M on dataset S.*

Note that the loss is computed on the validation set. The validation set can be disjoint from the training set to avoid overfitting or it can be the same as S_{train} to save some labeling budget.

Connection to meta-learning. The above problem definition is closely related to *meta-learning* which is the machine learning paradigm of "learning-to-learn". See [84] for a detailed survey. A meta-learning algorithm typically attempts to better learn a target task by leveraging external knowledge accumulated from the "experience" of learning similar tasks or domains. Such knowledge can be, for example, how to better initialize [23] or update [76] the model. For the DA optimization problem, we aim to learn a target model M by meta-learning the optimal *policy model for selecting and weighting augmented examples*. The meta-objective, as defined above, is to minimize the validation loss, which allows the policy model to keep improving itself by learning from the experience of "teaching" the target model with the augmented examples.

4.1 The filtering and weighting models

Rotom leverages a filtering model and a weighting model to select and assemble the training data to train the target model. Intuitively, the filtering model is a binary classifier that decides whether to take or discard an augmented example. The weighting model then determines the importance of the selected examples by assigning their weights for computing the training loss.

Definition 4.1 (Filtering and Weighting). For a sequence classification task with label vocabulary V, let $e = (x, \hat{x}, y)$ be an augmented example where x is an original sequence, $\hat{x}$ is an augmented sequence and $y \in V$ is the class label of x.

- A filtering model M_F is a binary classifier that takes as input an augmented example e and outputs $M_F(e) \in \{0, 1\}$.
- A weighting model M_W is a regression model that takes an augmented example e as input and outputs weight $M_W(e) \in [0, 1]$.

Why two models? Note that in principle, one can keep only the weighting model M_W and apply it to all the augmented examples. However, this is infeasible due to the very large amount of augmented data. For example, the number of augmented examples generated by the token-level transformations is at least proportional to the training set size times the sequence length. The number of sequences generated by InvDA can be even arbitrarily large. The filtering model acts as a coarse-grained pre-filter to quickly removes examples that are not desirable. The removed examples thus will not be processed by the weighting model or the target model to save computation resources.

The filtering model. For the above reason, we design the filtering model M_F to be a lightweight, feature-based, single-layer perceptron model. The input features of M_F consist of the one-hot encoded label y and the element-wise KL-divergence from the target model's predicted label distribution on x to the predicted label distribution on $\hat{x}$. Formally, recall that M is the target model and we denote by $p_M(x)$ the probability distribution obtained by applying M on x. Let $W_F \in \mathbb{R}^{2|V|\times 2}$ and $b_F \in \mathbb{R}^2$ be the trainable weight and bias of the perceptron model (here we overload the notation W_F to also denote the parameters of W_F). The filtering model computes

$$M_F(x, \hat{x}, y) = \mathsf{softmax}\big(W_F \times \mathsf{concat}\big(y, p_M(x) \log \frac{p_M(x)}{p_M(\hat{x})}\big) + b_F\big).$$

Intuitively, by using the KL-divergence as M_F's features, we expect the model to learn to remove augmented sequence $\hat{x}$ that is too different from the original sequence x according to the current model's prediction. Such sequences are likely to contain too many changes so that its semantics drifts from the original semantics. Moreover, adding the one-hot encoded y to the features allows the filtering model to calibrate the strength of the filter for each class (i.e., for cases where one class is noisier than the others).

The weighting model. After we remove the unwanted examples, we are ready to form batches to train the target model. Given a batch B of augmented examples, the goal of the weighting model

M_W is to adjust the weights of the examples so that the target model can be better optimized. Formally, to update the target model M, we compute the weighted training loss and back-propagate:

$$\text{Loss}_{\text{train}} = \sum_{(x,\hat{x},y)\in B} \text{CE}(y, p_M(\hat{x})) \times M_W(x, \hat{x}, y) \quad (1)$$

where $\text{CE}(\cdot)$ is the cross-entropy loss function.

The problem of assigning weights is not trivial. For example, we expect the weighting model to assign higher weights to the hard examples so that they are more likely to be captured by the target model. However, selecting features for the weighting model is much more complex than the filtering model because deciding the "hardness" requires a lot of sequence understanding capability. Thus, instead of manually engineering the features, we leverage pre-trained LMs to encode an augmented sequence such that our weighting model can access the rich features learned from pre-training. Specifically, M_W consists of a language model LM_W and a single linear layer L_W (whose input size is the same as the hidden size of LM_W and output size is 1). We compute M_W as

$$M_W(x, \hat{x}, y) = \text{sigmoid}\big(L_W(\text{LM}_W(\hat{x}))\big) + \|p_M(\hat{x}) - y\|_2. \quad (2)$$

Note that the output of M_W can be greater than 1 so we need to normalize the weights within the batch. The original sequence x is not used in the computation to save half of the computation.

There is a special additive term $\|p_M(\hat{x}) - y\|_2$ which computes the L2 distance between the predicted probability on the augmented $\hat{x}$ and the true label distribution y. The purpose of adding this term is to allow the weighting model to function normally even when it has not yet arrived at a stable state. At the beginning of the training process, the L2 distance would dominate the LM_W term because the target model M is uncertain on most of the examples. At this stage, M_W would simply assign higher weights to those examples that are more uncertain which is in analogy to the uncertainty-based sampling technique in active learning [77]. After a while when the model M fits most of the examples, the L2 distance is expected to get close to 0. We also expect the LM_W term to function normally since it has been trained simultaneously with M (as we shall describe next). Note that gradients are not propagated through the L2 distance term when we update M.

Our goal of optimizing the filtering model and the weighting model is related but not exactly the same as the classic concept of "learning-to-learn" where the *meta-learner* learns how to initialize or update the *learner* or the target model. In our setting, the policy model is the meta-learner and its meta-objective is to minimize the validation loss. Instead of learning to update the target model's parameters, we learn how to better prepare the target model's input by selecting and weighting the augmented data. Using the meta-training algorithm described next, we jointly optimize the policy and the target models (i.e., the meta-learner and the learner) such that the target model keeps improving as measured by the training loss while the policy model learns how to better select and weight the augmented examples as measured by the validation loss.

4.2 The meta-training algorithm

Algorithm 2 shows how Rotom simultaneously trains the target model, the filtering model, and the weighting model. The training is done in a two-phase manner that alternatively updates the target model and the filtering/weight models. We note that this algorithmic pattern is commonly seen in Reinforcement Learning [16, 32, 63, 70] and Automated Machine Learning [28, 48, 50, 52].

Algorithm 2: Meta-learning for Data Augmentation

input : A training set S_{train} of augmented examples; a validation set S_{val} ; the learning rate η
output : The target model M
variables : The filtering and weighting model, M_F and M_W; the training and validation loss, $\text{Loss}_{\text{train}}$ and Loss_{val}

1 Initialize M, M_F, and M_W with pre-trained or random weights;
2 **while** $\text{Loss}_{\text{train}}$ *does not converge* **do**
  /* Get a fresh train batch and a validation batch */
3 $B_{\text{train}} \leftarrow$ get_next_batch(S_{train});
4 $B_{\text{val}} \leftarrow$ get_next_batch(S_{val});
  /* **Phase 1: update the target model** */
5 $B_{\text{train}} \leftarrow \{e | e \in B_{\text{train}}$ and $M_F(e) = 1\}$; // Filter
  /* Compute the weighted loss using Eq. 1 */
6 $\text{Loss}_{\text{train}} = \sum_{(x,\hat{x},y)\in B_{\text{train}}} \text{CE}(y, p_M(\hat{x})) \times M_W(x, \hat{x}, y)$;
7 Update M with gradient $\nabla_M(\text{Loss}_{\text{train}})$;
  /* **Phase 2: update M_F and M_W** */
  /* A virtual gradient descent step */
8 $M' \leftarrow M - \eta \cdot \nabla_M(\text{Loss}_{\text{train}})$;
9 $\text{Loss}_{\text{val}} = \sum_{(x,y)\in B_{\text{val}}} \text{CE}(y, p_{M'}(x))$;
10 Update M_F with gradient $\nabla_{M_F}(\text{Loss}_{\text{val}})$; // Eq. 3
11 Update M_W with gradient $\nabla_{M_W}(\text{Loss}_{\text{val}})$; // Eq. 4
12 **return** M;

In the first phase (line 5-7), we apply the filtering model to assemble the training batch B_{train}. Note that in practice, B_{train} might be too small because of too aggressive filtering or contain multiple sequences generated from the same original example. We avoid such cases by properly sampling the training set S_{train} and refill the batch when some augmented examples get dropped. We then compute the weights with the weighting model M_W and the weighted training loss ($\text{Loss}_{\text{train}}$) following Eq. 1. Then the loss is back-propagated to update the target model M. We denote by $\nabla_M(\text{Loss})$ the gradient of a tensor variable Loss with respect to the parameters of M.

In the second phase (line 8-11), we perform a "virtual" gradient descent step to get M' by updating M with the gradient $\nabla_M \text{Loss}_{\text{train}}$. So M' is the updated M using the current filtering and weighting models M_F and M_W. Note that (in line 8) we overload the notation M to denote the parameters of M. The performance of M_F and M_W is measured by the validation loss Loss_{val} because intuitively we can expect Loss_{val} to descend fast if we train M with the right data and weights. Thus, we improve M_F and M_W by updating them (line 10-11) with the gradients of Loss_{val} with respect to their parameters.

Gradient estimation. The main challenge arises from computing the two gradients $\nabla_{M_F}(\text{Loss}_{\text{val}})$ and $\nabla_{M_W}(\text{Loss}_{\text{val}})$. The gradient $\nabla_{M_F}(\text{Loss}_{\text{val}})$ cannot be computed by a direct back-propagation because line 5 where we apply the filtering model is not differentiable. Instead, we estimate $\nabla_{M_F}(\text{Loss}_{\text{val}})$ using the REINFORCE estimator [87] which is also commonly used in policy-based Reinforcement Learning. Formally, we estimate the gradient by

$$\hat{\nabla}_{M_F}(\text{Loss}_{\text{val}}) = \nabla_{M_F}\Big(\text{Loss}_{\text{val}} \cdot \sum_{e\in B_{\text{train}}} \log\big(\text{prob}(M_F(e) = 1)\big)\Big) \quad (3)$$

where Loss_{val} is treated as a constant with no gradient computed and $\text{prob}(M_F(e) = 1)$ is the probability (as computed by M_F) of an example e passes the filter. We sum up the log-probability of all the augmented examples to get the log-probability of forming the resulting batch. Note that here we relax the deterministic output $M_F(e)$ to be a random variable drawn from the distribution $p_{M_F}(e)$ to better optimize M_F with explore-and-exploit [87]. This estimation method is less accurate because it ignores a large part of the intermediate computation. Nevertheless, it adds only minimal computation overhead so it aligns well with our goal of keeping the filtering model small and efficient.

Computing $\nabla_{M_W}(\text{Loss}_{\text{val}})$ is hard because a direct back-propagation requires computing a 2nd-order gradient (by the chain rule)

$$\nabla_{M_W}(\text{Loss}_{\text{val}}) = -\eta \nabla^2_{M_W,M} \text{Loss}_{\text{train}} \nabla_{M'} \text{Loss}_{\text{val}}$$

where Loss_{val} is a function over parameters of M' and $\text{Loss}_{\text{train}}$ is a function over the parameters of M_W and M. A full computation of this gradient can take time $O(|M| \cdot |M_W|)$ which is not practical. Instead, we follow [23, 52] to approximate the 2nd-order gradient using finite difference. Formally, let ϵ be a small constant (e.g., 0.01) and $M^{\pm} = M \pm \epsilon \nabla_{M'} \text{Loss}_{\text{val}}$. We estimate $\nabla_{M_W}(\text{Loss}_{\text{val}})$ by

$$\hat{\nabla}_{M_W}(\text{Loss}_{\text{val}}) = - \eta \frac{\nabla_{M_W}(\text{Loss}_{\text{train}}(M^+, M_W)) - \nabla_{M_W}(\text{Loss}_{\text{train}}(M^-, M_W))}{2\epsilon} \tag{4}$$

where $\text{Loss}_{\text{train}}(M^{\pm}, M_W)$ denotes the training loss computed using the modified target models $M^{\pm}$ and the same augmented examples' weights generated by M_W. This gradient estimation only requires first-order gradients thus can be computed efficiently in $O(|M| + |M_W|)$ time with 3 forward-backward passes. We refer the interested readers to [52] for more technical details.

5 ROTOM FOR SEMI-SUPERVISED LEARNING

Next, we show that Rotom's meta-learning framework can be naturally extended to semi-supervised learning (SSL) to leverage the large amount of unlabeled data. To this end, Rotom adopts *consistency regularization*, an SSL technique that was recently shown to be successful in computer vision and NLP tasks [5, 6, 79, 89].

Consistency regularization. Intuitively, consistency regularization leverages unlabeled data by encouraging the target model to make consistent predictions among noisy variants of an unlabeled example. For example, given an unlabeled product review *"The best book ever"*, without knowing its sentiment label, we expect the target model to predict its sentiment consistent with the prediction on a similar review *"The best movie ever"*.

Consistency regularization achieves this effect via (1) transformation of the original unlabeled data by DA operators and (2) label sharpening. Formally, given a set $U = \{x_1, \ldots, x_u\}$ of unlabeled sequences, a *consistency regularizer* of the following form is added to the original supervised training loss:

$$\sum_{x \in U} \text{Loss}\Big(\text{sharpen}\big(p_M(x)\big), p_M(\hat{x})\Big) \tag{5}$$

where Loss is a loss function (e.g., cross-entropy) and $\hat{x}$ is an augmented sequence of x. The $\text{sharpen}(\cdot)$ function takes the current model's prediction as input and converts it into a distribution close to being one-hot. The $\text{sharpen}(\cdot)$ term is treated as a constant tensor during gradient back-propagation. We consider two variants of the $\text{sharpen}(\cdot)$ function in Rotom:

$$\text{sharpen\_v1}(p, T)_i = p_i^{1/T} \Big/ \sum_{j=1}^{|V|} p_j^{1/T}, \text{ for } i \in [1, \ldots, |V|], \tag{6}$$

$$\text{sharpen\_v2}(p, \theta)_i = \mathbb{1}(p_i \geq \theta), \text{ for } i \in [1, \ldots, |V|]. \tag{7}$$

The first variant (Eq. 6) is used in [5, 6, 89] with a hyper-parameter T to control how close the output is to a one-hot distribution (the smaller the closer for $T \in (0, 1]$). The second variant (Eq. 7) is commonly known as *pseudo-labeling* [45, 79] which directly assigns x with the label $\arg\max(p_M(x))$ if the maximal confidence is above the threshold θ (for $\theta > 1/|V|$). Rotom combines guessed labels generated by both variants of sharpen.

Similar to DA, the main challenge in consistency regularization is that not all the unlabeled examples are equally helpful to the target model. For example, the guessed label $\text{sharpen}\big(p_M(x)\big)$ might be wrong because the target model M is not yet stable. The augmented example $\hat{x}$ might not preserve the label either. These are exactly cases where our meta-learning framework comes in handy.

Extending the meta-learning framework. Intuitively, Rotom incorporates unlabeled data by applying the weighting model M_W on *both the labeled and unlabeled* data. We simply need to modify Algorithm 2 as follows:

- Line 5: we add to the training batch B_{train} a batch of unlabeled examples $B_{\text{unlabeled}} \subseteq U$ of size equal to $|B_{\text{train}}|$;
- Line 6: when computing the weight of each example, replace y with the guessed label, $\text{sharpen}\big(p_M(x)\big)$.

Note that we do not apply the filtering model on the unlabeled data to avoid label imbalance. This is because the filtering model can remove significantly more unlabeled examples from one class than the others (possibly because the class has fewer labels) causing the less confident class to be further imbalanced.

One immediate advantage of applying the meta-learning framework is that Rotom can assign low weights to the unlabeled examples at the beginning of the training process and later increase the weights as the model is trained. This is typically a hard-coded strategy in existing SSL algorithms and requires tuning hyper-parameters (e.g., at what condition the model should start using the unlabeled data), but now this becomes fully automated in Rotom.

6 EXPERIMENTS

Next, we present the experiment results on benchmark datasets for Entity Matching (EM), Error Detection (EDT), and Text Classification (TextCLS). Our results show that Rotom effectively combines simple DA operators and the InvDA operator even when some of them do not yield performance gain when applied independently. With this strength, in the low-resource settings, Rotom outperforms the state-of-the-art (SOTA) EM systems by up to 6% F1 score and the SOTA error detection system by 7.64% of average F1 score. Rotom also outperforms two recent NLP data augmentation methods in TextCLS tasks. Our training time analysis shows that Rotom's meta-learning framework introduces an overhead to the baseline of LM fine-tuning much smaller than the hyper-parameter space.

6.1 Implementation and baselines

We implemented Rotom in PyTorch [65] and with the Transformers library [88]. We used the 12-layer RoBERTa [53] model in the EM and EDT experiments and used the uncased DistilBERT [74] and BERT [19] model in TextCLS. Across all experiments, we set the batch size to be 32, learning rate to be 3e-5 and the max sequence length to be 128. In each run, we fine-tune the LM (and components of Rotom) with the Adam optimizer for at most 40 epochs (or less if provided with more training data), select the checkpoint with the highest accuracy/F1 on the validation set, and report the test accuracy/F1. For each dataset, we repeat the experiment 5 times and report the average accuracy/F1. We ran all the experiments on a p3.8xlarge AWS EC2 machine with 4 V100 GPUs (1 GPU per run). We open-sourced Rotom at https://github.com/megagonlabs/rotom.

We evaluate and compare the following 5 methods and task-specific baselines described in corresponding subsections:

- **Baseline (RoBERTa/DistilBERT/BERT)**: This baseline method fine-tunes the pre-trained language model (LM) on the original training examples without any data augmentation.
- **MixDA**: This method applies a single DA operator listed in Table 3. We tune the DA operator as a hyper-parameter and select one operator that generally works well for each type of task (EM, EDT, or TextCLS). At every epoch, we obtain the augmented set by randomly transforming each training example with the operator and fine-tune the LM on the augmented set. The DA operator is applied with the MixDA technique [58] which interpolates the LM representation of an augmented example with the original one to "partially" apply the operator. This method was shown to be generally effective in multiple tasks including EM [49, 58].
- **InvDA**: This method augments with the seq2seq DA operator InvDA. To better understand the effect of InvDA, we follow the same training procedure for MixDA except replacing the DA operator with InvDA. Without hyper-parameter tuning, we train the InvDA model by fine-tuning the T5 [67] pre-trained model (see Section 3) on a set of unlabeled data for each task. For TextCLS, because of the similarity across tasks, we fine-tune a single InvDA model using unlabeled data from all the TextCLS tasks. We use top-k sampling with $k = 120$ over the top 98% most likely tokens [31] to generate at most 50 unique transformed sequences from each example. We randomly select one as the augmented sequence at each epoch.
- **Rotom**: This is the proposed meta-learning framework that selects and weights the augmented examples to better fine-tune the target model (See Section 4). The weighting model uses the same LM architecture as the target model. In our experiments, we use Rotom to combine the original training examples with the augmented examples generated by InvDA and MixDA.
- **Rotom+SSL**: Finally, we evaluate the semi-supervised variant of Rotom that selects and weights the unlabeled data to train the target model. For each dataset, Rotom+SSL uses at most 10,000 uniformly sampled unlabeled examples. We note that the performance can potentially be further improved if we provide more unlabeled data.

6.2 Datasets

We evaluate the above systems on standard benchmark datasets of EM, EDT, and TextCLS. We listed these datasets in Table 6 and 7.

The EM datasets are widely used in the literature [49, 61]. Each dataset consists of labeled pairs of product or publication records from different websites. The goal is to classify whether a pair of records represent the same product/publication. For each dataset, we uniformly sample the training set from the original train+valid set and we vary the sample size from 300 to 750. Three of the datasets have a dirty version available which we also use to evaluate robustness of each method against data noise such as misplaced attributes. To save the labeling budget, we do not create a new validation set but simply use the training set for validation. We measure the models' performance by the F1 scores.

The EDT datasets are from [55]. Each dataset consists of a dirty spreadsheet and the goal is to classify whether a cell contains an error or not. For each spreadsheet, we hold out 20 randomly sampled rows (so up to 400 cells) for evaluation. To train Rotom and the baseline methods, we construct a training set of size from 50 to 200 cells having the same number of clean and dirty cells (to avoid label imbalance). Note that because of the small sample size, we do not allocate labeling budget to create the validation set but just use the training set for validation. We use the context-independent serialization (See Section 2) since it achieves better results on these datasets. We measure the performance by the F1 scores for the EDT tasks.

For TextCLS, we use 8 standard NLP benchmark datasets. The datasets are for different purposes including sentiment analysis and intent classification. We vary the size of the training and validation set (uniformly sampled) from 100 to 500 to test the label efficiency (so the labeling budget is from 200 to 1k). We measure the performance by the classification accuracy.

For each sample of a dataset, we treat the remaining training examples as unlabeled for semi-supervised learning in Rotom+SSL.

Table 6: EM and EDT datasets. For each EM dataset, we create a sample of training/validation set of size from 300 to 750 and use the original test set for evaluation. Each EM dataset marked with a "∗" also comes with a more challenging dirty version. For the EDT datasets, we vary the training set size from 50 to 200 cells (no validation set). Each test set consists of 20 uniformly sampled tuples.

EM Dataset	#Train+ #Valid	#Test	EDT Dataset	Test size (#cell, #tpl)	Table size (#tpl)
Amazon-Google	9,167	2,293	beers	220 / 20	2,410
DBLP-ACM*	9,890	2,473	hospital	400 / 20	1,000
DBLP-Scholar*	22,965	5,742	movies	340 / 20	7,390
Walmart-Amazon*	8,193	2,049	rayyan	220 / 20	1,000
Abt-Buy	7,659	1,916	tax	300 / 20	200K

Table 7: TextCLS Datasets. For each dataset, we uniformly sample a training set and a validation set of size 100, 300, and 500 respectively. For the first 3 large datasets, we sample a test set of size 1,000 for more efficient evaluation. We use the original test sets otherwise.

Dataset	#classes	(#Train,#Test)	Sample Test?	Class semantics
AG	4	(120k, 7,600)	yes	News topic
AM-2	2	(3.6m, 400k)	yes	Amazon review sentiment
AM-5	5	(3m, 650k)	yes	Amazon review sentiment
SST-2	2	(6,920, 1,821)	no	Movie review sentiment
SST-5	5	(8,544, 2,210)	no	Movie review sentiment
TREC	6	(5,452, 500)	no	Open-domain question intent
ATIS	24	(4,478, 893)	no	Airline reservation intent
SNIPS	7	(13,084, 700)	no	Voice assistant intent

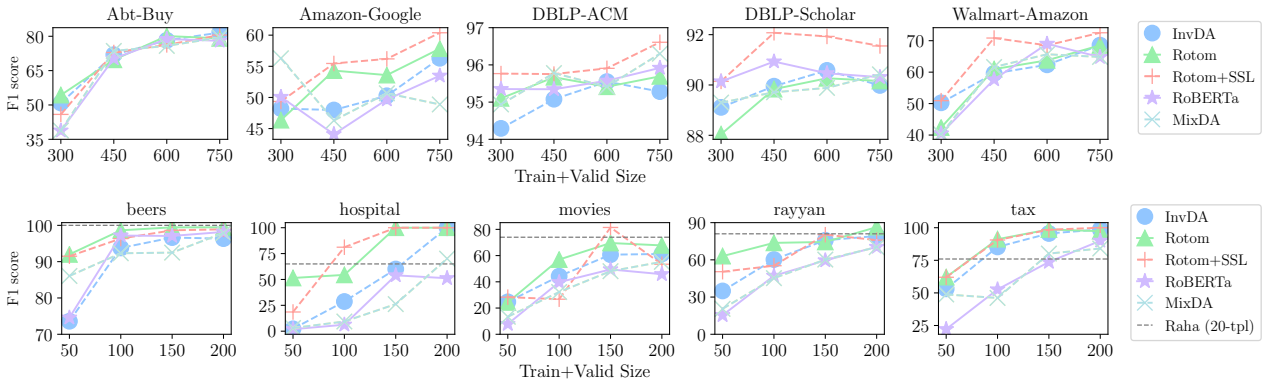

Figure 3: Varying the labeling budgets on the EM (upper) and EDT (lower) tasks. Compared to the baselines, Rotom+SSL or Rotom achieve the overall best performance in most cases across different labeling budgets.

Table 8: F1 scores on 5 EM datasets with at most 750 training+validation examples. The dataset names are shortened as "Abt-Buy" → "A-B" etc. The numbers for DeepMatcher are taken from [61] when the model is trained on the full datasets.

Method	A-B	A-G	D-A clean / dirty	D-S clean / dirty	W-A clean / dirty
DM (full, [61])	62.8	69.3	98.4 / 98.1	94.7 / 93.8	67.6 / 53.8
DM+RoBERTa	73.73	63.91	97.10 / 96.06	91.81 / 90.69	54.25 / 50.14
RoBERTa	78.03	53.48	95.92 / 96.11	90.31 / 91.59	64.80 / 59.69
Brunner et al.	76.63	52.33	96.09 / 96.57	91.21 / 90.76	66.48 / 59.97
MixDA	79.44	56.29	96.30 / 95.86	90.42 / 90.53	65.68 / 70.02
InvDA	**81.61**	56.29	95.55 / 95.99	90.58 / 90.56	68.59 / 69.64
Rotom	80.09	57.82	95.69 / 95.80	90.27 / 90.57	68.51 / **71.43**
Rotom+SSL	80.53	**60.38**	**96.61** / 95.98	**92.08 / 92.21**	**72.52** / 70.62

6.3 Main results on Entity Matching

We compare our methods with the RoBERTa baseline, MixDA, and the following two EM solutions:

(1) **DeepMatcher (DM) [61]**: DM is a classic deep learning EM method with many follow-up works [24, 34, 38, 49, 62, 91, 95]. Instead of fine-tuning LMs, DM trains a hybrid neural net consisting of RNN layers and the Attention mechanism. DM achieves good results in multiple EM tasks but requires a significant amount of (~10k) training data. The reported numbers are taken directly from [61] which are DM's F1 scores when it is trained on the full datasets. For fair comparison, we also consider a variant of DM with its RNN and word embedding layers replaced by a RoBERTa encoder. We denote this variant as DM+RoBERTa.

(2) **Brunner et al. [9]** for EM: In [9], Brunner et al. propose a deep EM solution based on LMs including BERT and RoBERTa. The model architecture is similar to Ditto but uses a different method to serialize entity records into LMs' input format. We consider the RoBERTa variant of their method for a fair comparison.

Table 8 shows the F1 scores of each method using at most 750 training/validation examples. Compared to DM trained on the full datasets, Rotom+SSL achieves an 1.9% higher average F1 (80.4 vs. 78.5) using only 6.5% of labels. This result indicates a significant potential saving of labeling effort in creating high-quality EM solutions. Compared to the RoBERTa baseline, both regular DA (MixDA) and InvDA effectively improve the models' performance. InvDA has significant effect on the A-B and W-A datasets by improving the baseline performance by over 3%. The results also show that Rotom+SSL effectively combines regular DA, InvDA, and the unlabeled data as Rotom+SSL outperforms *both* MixDA and InvDA in 7/8 cases by up to 6.04% (W-A). Semi-supervised learning has a large impact for EM as Rotom+SSL consistently outperforms Rotom in 7/8 cases. The other two baseline methods, DM+RoBERTa and Brunner et al. [9], do not show a significant overall difference (<0.4 average F1) with RoBERTa. This minor difference justifies choosing RoBERTa as the base version of Rotom for EM. Among all three datasets that come with dirty versions, we do not observe noticeable changes by comparing the clean/dirty F1 scores, and Rotom can achieve a non-trivial improvement on W-A compared to baseline methods. This result confirms that Rotom is as robust as the baselines against data noise.

Table 9: F1 scores on 5 error detection tasks. We reported the results of Raha w. 20 labeled tuples and the results of Rotom (and its baselines) w. at most 200 labeled cells (<20 tuples, Table 6).

Method	beers	hospital	movies	rayyan	tax	AVG
Raha (20-tpl, [55])	**100.0**	74.29	80.80	76.26	91.11	84.49
RoBERTa	98.19	53.98	49.45	70.43	89.97	72.40 (-12.09)
MixDA	97.91	70.12	55.25	70.22	84.05	75.51 (-8.98)
InvDA	96.63	**100.0**	61.26	79.21	98.67	87.15 (+2.66)
Rotom	99.45	**100.0**	69.59	**86.38**	98.67	90.82 (+6.33)
Rotom+SSL	98.90	**100.0**	**81.44**	80.29	**100.0**	92.13 (+7.64)

6.4 Main results on data cleaning

We compare our methods with **Raha [55]** and the RoBERTa baseline for error detection. Raha is the SOTA error detection system based on ensemble learning. Raha achieves high label efficiency via interactive labeling of the clean/dirty tuples while Raha ensembles multiple error detection models. According to [55], Raha achieves high F1 scores with only 20 tuples labeled. We use Raha's open-source version in our experiments.

Table 9 shows the performance of Rotom and Rotom+SSL when using no more than 200 labeled cells. Compared to the SOTA error detection system Raha, Rotom+SSL outperforms in 4/5 datasets and

by an average of 7.64% while using strictly fewer labeled cells. For example, on the tax dataset, Rotom achieves a perfect F1 score (9% higher than Raha) with 200 cells, which is only 2/3 of the labels provided to Raha. The hospital dataset is in the same situation with only 1/2 of the labeled cells of Raha. We also found that InvDA is more effective than simple DA operators (>10% on average). This is because simple transformations such as word deletion or shuffling are likely to corrupt cells that are originally clean. InvDA produces augmented examples that are more likely to be label-preserving. Recall that Rotom takes in augmented examples generated by simple DA (dirty) and InvDA (clean). It is interesting that the overall F1 still improves significantly (by >3%). Similarly, we observed that Rotom consistently improves the performance of InvDA with different labeling budgets across multiple datasets (Figure 3). This result shows that the meta-learning framework can select the right examples for training to amplify the effectiveness of DA.

6.5 Main results on text classification

Table 10 shows the results of Rotom on the TextCLS datasets. Rotom and Rotom+SSL show significant improvement when the labeling budget is small. When the training/validation set size is 100, Rotom consistently outperforms the DistilBERT baseline by 8.59% F1 on average and up to 20% F1 (TREC). Note that MixDA is more effective than InvDA for the TextCLS tasks which implies that increasing the diversity of augmented examples does not help in general. MixDA has a positive effect on 14/24 settings while InvDA is positive at only 11/24 settings. Still, Rotom can combine them for better overall performance achieving 20/24 improved results.

We compare in Table 11 the results of Rotom with the BERT baseline and two SOTA data augmentation techniques, **Hu et al. [32] and Kumar et al. [44]**. Both methods perform well on a low labeling budget. Hu et al. [32] train a DA operator and a weighting policy via reinforcement learning. Kumar et al. leverage LMs such as GPT-2, BERT, or BART to generate the augmented sequences. Note that the numbers are different from Table 10 because we change our experiment settings to follow the exact settings in [32] and [44]: (1) Hu et al. samples 40 training examples per class and (2) Kumar et al. uniformly samples 1% of the training set. Both methods sample 5 examples per class for validation. IMDB is another binary sentiment classification dataset that we exclude from our main result[1].

These two methods are technically close to Rotom since (1) Hu et al. learns a DA operator and a weighting policy separately via reinforcement learning and (2) Kumar et al. fine-tunes LMs to generate augmented examples via *conditional generation*. Across the 6 cases, Rotom outperforms the two methods on absolute accuracy in 5/6 settings and by up to 25% (vs. Kumar et al. on TREC). Moreover, Rotom achieves higher relative improvement to the BERT baseline in 4/6 cases. The performance difference can be explained by:

- Compared to Hu et al., Rotom uses InvDA which can generate diverse augmented examples while the DA operator learned by Hu et al. modifies at most 1 token.
- Compared to Kumar et al., Rotom additionally filters/weights the potentially noisy augmented examples from the generative LMs.

[1]IMDB consists of sequences much longer than the max length of 128 which explains the low overall accuracy in Table 11.

Table 10: Accuracy of Rotom on 8 TextCLS datasets.

	Size	AG	AM-2	AM-5	ATIS	SNIPS	SST-2	SST-5	TREC	AVG
Distil-BERT	100	72.44	69.88	26.60	73.35	84.14	71.75	28.81	65.56	60.01
	300	76.42	77.78	36.72	87.10	93.89	83.93	37.89	90.68	72.57
	500	79.16	81.30	40.72	91.27	95.03	85.99	42.38	94.48	75.88
MixDA	100	71.76↓	71.44↑	27.56↑	73.95↑	86.94↑	69.96↓	28.67↓	67.04↑	60.80(+0.78)
	300	75.62↓	78.42↑	34.54↓	86.05↓	94.03↑	84.10↑	39.00↑	90.36↓	72.36(-0.21)
	500	78.94↓	81.98↑	40.22↓	91.60↑	95.20↑	86.12↑	43.72↑	93.32↓	76.02(+0.14)
InvDA	100	72.92↑	70.76↑	25.96↓	72.39↓	86.43↑	73.50↑	29.73↑	63.64↓	60.34(+0.33)
	300	75.46↓	77.60↓	36.86↑	85.40↓	94.06↑	84.92↑	38.37↑	89.88↓	72.44(-0.13)
	500	78.60↓	80.20↓	40.46↓	91.11↓	95.23↑	85.29↓	44.73↑	94.08↓	75.87(-0.01)
Rotom	100	74.20↑	75.74↑	31.24↑	84.01↑	92.17↑	76.63↑	30.83↑	82.76↑	67.63(+7.61)
	300	75.82↓	76.50↓	36.82↑	90.03↑	94.43↑	84.27↑	38.46↑	88.60↓	72.73(+0.16)
	500	78.16↓	80.00↓	40.22↓	93.95↑	95.74↑	85.35↓	41.99↓	89.60↓	75.27(-0.61)
Rotom +SSL	100	76.18↑	76.88↑	30.40↑	82.87↑	93.20↑	78.35↑	32.93↑	85.60↑	68.60(+8.59)
	300	77.66↑	78.48↑	34.85↓	90.26↑	94.91↑	85.25↑	40.99↑	91.24↑	73.71(+1.14)
	500	79.34↑	80.33↓	39.60↓	92.43↑	95.81↑	86.27↑	44.24↑	94.00↓	76.10(+0.22)

Table 11: Comparing the classification accuracy of Rotom with Hu et. al 19 [32] and Kumar et al. 20 [44]. We highlight the difference with the BERT baseline in green/red.

	IMDB	SST-5	TREC		SNIPS	SST-2	TREC
BERT	64.90	33.47	87.76	BERT	90.77	65.00	38.12
MixDA	66.90	34.38	87.16	MixDA	90.46	67.62	39.16
	+2.00	+0.91	-0.60		-0.31	+2.62	+1.04
InvDA	65.98	35.71	86.64	InvDA	91.69	64.59	30.76
	+1.08	+2.24	-1.12		+0.91	-0.41	-7.36
Rotom	72.48	37.36	88.04	Rotom	92.37	73.75	62.72
	+7.58	+3.89	+0.28		+1.60	+8.75	+24.60
Hu et al. 19				Kumar et al. 20			
BERT	63.55	33.32	88.25	BERT	57.95	59.08	30.65
+Learned DA	65.62	37.03	89.15	+CG w. BART	81.68	63.00	37.48
	+2.07	+3.71	+0.90		+23.73	+3.92	+6.83
+Weighting	64.78	36.51	89.01	+CG w. BERT	81.31	61.90	31.88
	+1.23	+3.19	+0.76		+23.36	+2.82	+1.23

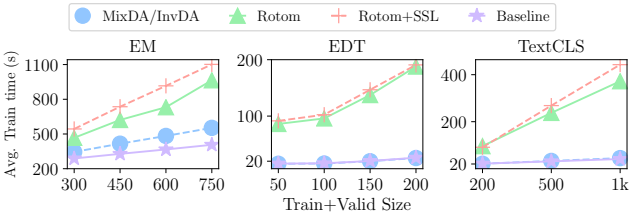

Figure 4: Rotom's training time for each domain (EM, EDT, and TextCLS). Each point is the average over all the datasets of a domain. The longest training time is 18.36 minutes (EM w. 750 examples). Rotom takes on average 5.6x of the InvDA/MixDA time to train.

6.6 Training time

Unlike other DA solutions based on Reinforcement Learning which can take tens to thousands of GPU hours to train [16, 28, 51], Rotom can be trained in minutes. Figure 4 shows the training time of Rotom compared to the baseline LM fine-tuning and MixDA. The training time for InvDA overlaps with MixDA since we pre-compute and cache the sequences generated by the seq2seq model. Rotom can be trained in <20 minutes in all cases. The overhead compared to the baseline DA method is only 5.6x on average and up to 9.8x. This is much smaller than the search space of DA operators. For example, enumerating all combinations of 2 token-/span-level operators in Table 3 incurs a 22x overhead. In addition, we observe that Rotom+SSL does not add significant overhead to Rotom as the difference is within 30% of Rotom's training time.

7 RELATED WORK

Entity matching (EM). EM has been studied extensively in data management, data mining, and NLP communities [25, 37] over the last 20 years. One of the key challenges in EM [37] is the pairwise matching problem, which determines if a given pair of data entries refer to the same real-world entity. Many pairwise matching methods treat the problem as a binary classification problem and solve the problem using machine learning models, such as decision trees [81], SVM [11], and CRF [56]. Recent solutions [9, 18, 34, 49, 61, 95] adapt more advanced machine learning models, in particular, deep learning models [34, 61] and pre-trained models [9, 49, 95], to further improve the performance. As training a highly accurate classification model often requires a substantial number of labeled examples, people have also investigated active learning to reduce the human annotation cost [3, 4, 18, 57, 75].

Error detection (EDT). EDT focuses on identifying erroneous values in a dataset and it is the first step in the data cleaning process [13]. Qualitative data cleaning [1, 7, 36, 71] employs rules or integrity constraints to detect the erroneous data. Quantitative data cleaning [30, 39], or outlier detection, on the other hand, focuses on conducting statistical analysis to identify data values showing abnormal distribution. Like EM, a new trend that using machine learning models to identify erroneous values has emerged. ActiveClean [40] dynamically trains a data cleaning model by interactively collecting new labels from users; Raha [55] tries to improve the training process by using an optimized representative value selection/sampling technique; HoloDetect [29] enriches the training dataset via a sophisticated task-specific data augmentation algorithm. Likewise, these machine learning-based techniques also employ a variety of optimizations to minimize the amount of training data. Leveraging machine learning while reducing human annotation efforts also motivate Rotom.

Different from existing ML-based EM and EDT solutions that focus on a single downstream task, Rotom is a versatile framework applicable to a range of data management and mining tasks that can be formulated as sequence classification.

Data augmentation (DA). Data augmentation (DA) has received increasing attention recently in machine learning problems across fields [16, 29, 41, 44, 86]. Commonly used generic simple DA operators for text data include word replacement [21, 35, 86, 94], word insertion/deletion/swapping [86], and back translation [90]. A major limitation of simple DA operators is that the generated examples are lack of diversity. This is because such operators can only perform a few local transformations over existing training examples. To overcome this limitation, recent works [2, 44] start using text generation models (a.k.a. seq2seq model) to produce more diverse examples conditioned on the given label. However, such generation-based DA may over-diversify the augmented examples and they are hard to train when the training set is too small, especially when the vocabulary of class labels is large. Rotom further enriches the DA family by introducing a novel DA operator InvDA. InvDA learns to augment existing examples in a self-supervised manner, and hence it can greatly reduce over-diversified examples and is loosely restricted by the original training data sizes.

Automatic data augmentation and meta-learning. There has been research on automating the process of discovering effective DA policies. These techniques formulate DA as a learning task of different optimization goals and solve them with different search strategies. The most popular optimization objective is to minimize the validation loss [16, 17, 32, 48, 50, 52, 63]. Other objectives include maximizing the similarity between distributions of the augmented/unaugmented data [28, 51] and customized generative adversarial objective [70]. Given an objective, to optimize a model architecture, existing searching techniques often use Bayesian optimization [51], reinforcement learning [16, 32, 63, 70], and meta-learning [28, 48, 50, 52]. Among these approaches, meta-learning-based searching techniques show better efficiency since they use gradient descent by differentiating the search space. In this paper, Rotom adapts the most popular optimization objective (minimizing the validation loss) and the more efficient meta-learning-based searching technique to select and combine augmented examples.

Training data collection in the DB community. To address the needs for more training examples, the DB community has developed lines of work on data collection including data programming [69, 82], crowdsourcing [26, 64], dataset discovery [22, 27], and integrating data from multiple sources [72]. More recently, under the setting where the dataset has multiple tables, researchers have investigated whether joining tables can improve the performance of the trained model [10, 43, 78]. Although sometimes also referred to as data augmentation [10], these techniques aim at adding effective features to each training example which is orthogonal to Rotom's goal of adding more effective training examples. These techniques can potentially be applied together with Rotom to achieve better performance.

8 CONCLUSION AND DISCUSSION

We introduced Rotom, a meta-learned data augmentation framework for data management and mining tasks. Rotom first leverages multiple data augmentation techniques including an innovative seq2seq-based operator InvDA. Next, Rotom adopts a meta-learning framework to train policies for selecting and weighting augmented examples jointly with the target model, thus it addresses the trade-off between diversity and quality in data augmentation. Our results show that the versatility of Rotom enables it to effectively combine augmented examples by simple DA operators and InvDA as well as unlabeled examples while training separately on these examples does not improve the model's performance; hence it outperforms previous methods on all three tasks with limited labeled data.

Rotom is closely related to the line of research of training data debugging/cleaning with techniques developed based on constraints [14, 68], human inputs [15, 40], or machine learning models [20, 29, 55]. The idea of filtering and re-weighting the noisy training examples would be directly applicable. Moreover, instead of relying on static rules or ML models to separately clean training data, one can apply Rotom's principle of jointly training the cleaning operators with the target model. We believe this is a promising direction for designing a new data cleaning pipeline that is user-friendly and effective for downstream ML tasks.

Acknowledgements. We are grateful to Wang-Chiew Tan for the insightful discussions and early support which tremendously help improve the quality of this work. The first author is partly supported by NSF grants IIS-1552538 and IIS-2008107.

REFERENCES

[1] Ziawasch Abedjan, Cuneyt G Akcora, Mourad Ouzzani, Paolo Papotti, and Michael Stonebraker. 2015. Temporal rules discovery for web data cleaning. *PVLDB* 9, 4 (2015), 336–347.

[2] Ateret Anaby-Tavor, Boaz Carmeli, Esther Goldbraich, Amir Kantor, George Kour, Segev Shlomov, Naama Tepper, and Naama Zwerdling. 2020. Do Not Have Enough Data? Deep Learning to the Rescue!. In *AAAI*. 7383–7390.

[3] Arvind Arasu, Michaela Götz, and Raghav Kaushik. 2010. On active learning of record matching packages. In *SIGMOD*. 783–794.

[4] Kedar Bellare, Suresh Iyengar, Aditya G Parameswaran, and Vibhor Rastogi. 2012. Active sampling for entity matching. In *KDD*. 1131–1139.

[5] David Berthelot, Nicholas Carlini, Ekin D Cubuk, Alex Kurakin, Kihyuk Sohn, Han Zhang, and Colin Raffel. 2019. Remixmatch: Semi-supervised learning with distribution matching and augmentation anchoring. In *ICLR*.

[6] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. 2019. Mixmatch: A holistic approach to semi-supervised learning. In *NeurIPS*. 5049–5059.

[7] Philip Bohannon, Wenfei Fan, Michael Flaster, and Rajeev Rastogi. 2005. A cost-based model and effective heuristic for repairing constraints by value modification. In *SIGMOD*. 143–154.

[8] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165* (2020).

[9] Ursin Brunner and Kurt Stockinger. 2020. Entity Matching with Transformer Architectures - A Step Forward in Data Integration. In *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, Angela Bonifati, Yongluan Zhou, Marcos Antonio Vaz Salles, Alexander Böhm, Dan Olteanu, George H. L. Fletcher, Arijit Khan, and Bin Yang (Eds.). OpenProceedings.org, 463–473. https://doi.org/10.5441/002/edbt.2020.58

[10] Nadiia Chepurko, Ryan Marcus, Emanuel Zgraggen, Raul Castro Fernandez, Tim Kraska, and David Karger. 2020. ARDA: Automatic Relational Data Augmentation for Machine Learning. *PVLDB* 13, 9 (2020), 1373–1387.

[11] Peter Christen. 2008. Automatic record linkage using seeded nearest neighbour and support vector machine classification. In *KDD*. 151–159.

[12] Peter Christen. 2012. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer.

[13] Xu Chu and Ihab F Ilyas. 2016. Qualitative data cleaning. *PVLDB* 9, 13 (2016), 1605–1608.

[14] Xu Chu, Ihab F. Ilyas, and Paolo Papotti. 2013. Holistic data cleaning: Putting violations into context. In *29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013*, Christian S. Jensen, Christopher M. Jermaine, and Xiaofang Zhou (Eds.). IEEE Computer Society, 458–469.

[15] Xu Chu, John Morcos, Ihab F. Ilyas, Mourad Ouzzani, Paolo Papotti, Nan Tang, and Yin Ye. 2015. KATARA: A Data Cleaning System Powered by Knowledge Bases and Crowdsourcing. In *SIGMOD*, Timos K. Sellis, Susan B. Davidson, and Zachary G. Ives (Eds.). ACM, 1247–1261.

[16] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. 2019. Autoaugment: Learning augmentation strategies from data. In *CVPR*. 113–123.

[17] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. 2020. Randaugment: Practical automated data augmentation with a reduced search space. In *CVPR workshops*. 702–703.

[18] Sanjib Das, Paul Suganthan GC, AnHai Doan, Jeffrey F Naughton, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, Vijay Raghavendra, and Youngchoon Park. 2017. Falcon: Scaling up hands-off crowdsourced entity matching to build cloud services. In *SIGMOD*. 1431–1446.

[19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*. 4171–4186.

[20] Mohamad Dolatshah, Mathew Teoh, Jiannan Wang, and Jian Pei. 2018. Cleaning Crowdsourced Labels Using Oracles For Statistical Classification. *PVLDB* 12, 4 (2018), 376–389.

[21] Marzieh Fadaee, Arianna Bisazza, and Christof Monz. 2017. Data augmentation for low-resource neural machine translation. *arXiv preprint arXiv:1705.00440* (2017).

[22] Raul Castro Fernandez, Ziawasch Abedjan, Famien Koko, Gina Yuan, Samuel Madden, and Michael Stonebraker. 2018. Aurum: A Data Discovery System. In *ICDE*. IEEE Computer Society, 1001–1012.

[23] Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *ICML*, Vol. 70. 1126–1135.

[24] Cheng Fu, Xianpei Han, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. 2019. End-to-end multi-perspective matching for entity resolution. In *AAAI*. AAAI Press, 4961–4967.

[25] Lise Getoor and Ashwin Machanavajjhala. 2012. Entity resolution: theory, practice & open challenges. *PVLDB* 5, 12 (2012), 2018–2019.

[26] Daniel Haas, Jiannan Wang, Eugene Wu, and Michael J. Franklin. 2015. CLAMShell: Speeding up Crowds for Low-latency Data Labeling. *PVLDB* 9, 4 (2015), 372–383.

[27] Alon Y. Halevy, Flip Korn, Natalya Fridman Noy, Christopher Olston, Neoklis Polyzotis, Sudip Roy, and Steven Euijong Whang. 2016. Goods: Organizing Google's Datasets. In *SIGMOD*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 795–806.

[28] Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. 2019. Faster autoaugment: Learning augmentation strategies using backpropagation. *arXiv preprint arXiv:1911.06987* (2019).

[29] Alireza Heidari, Joshua McGrath, Ihab F Ilyas, and Theodoros Rekatsinas. 2019. Holodetect: Few-shot learning for error detection. In *SIGMOD*. 829–846.

[30] Joseph M Hellerstein. 2008. Quantitative data cleaning for large databases. *United Nations Economic Commission for Europe (UNECE)* 25 (2008).

[31] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The Curious Case of Neural Text Degeneration. In *ICLR*. OpenReview.net.

[32] Zhiting Hu, Bowen Tan, Russ Salakhutdinov, Tom Mitchell, and Eric Xing. 2019. Learning data manipulation for augmentation and weighting. In *NeurIPS*. 15764–15775.

[33] Ihab F. Ilyas and Xu Chu. 2019. *Data Cleaning*. ACM.

[34] Jungo Kasai, Kun Qian, Sairam Gurajada, Yunyao Li, and Lucian Popa. 2019. Low-resource deep entity resolution with transfer and active learning. *arXiv preprint arXiv:1906.08042* (2019).

[35] Sosuke Kobayashi. 2018. Contextual augmentation: Data augmentation by words with paradigmatic relations. *arXiv preprint arXiv:1805.06201* (2018).

[36] Solmaz Kolahi and Laks VS Lakshmanan. 2009. On approximating optimum repairs for functional dependency violations. In *ICDT*. 53–62.

[37] Pradap Konda, Sanjib Das, Paul Suganthan GC, AnHai Doan, Adel Ardalan, Jeffrey R Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeff Naughton, et al. 2016. Magellan: Toward building entity matching management systems. *PVLDB* 9, 12 (2016), 1197–1208.

[38] Ioannis Koumarelas, Thorsten Papenbrock, and Felix Naumann. 2020. MDedup: duplicate detection with matching dependencies. *PVLDB* 13, 5 (2020), 712–725.

[39] Hans-Peter Kriegel, Peer Kröger, and Arthur Zimek. 2010. Outlier detection techniques. *Tutorial at KDD* 10 (2010), 1–76.

[40] Sanjay Krishnan, Jiannan Wang, Eugene Wu, Michael J Franklin, and Ken Goldberg. 2016. Activeclean: Interactive data cleaning for statistical modeling. *PVLDB* 9, 12 (2016), 948–959.

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NeurIPS*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105.

[42] Arun Kumar, Matthias Boehm, and Jun Yang. 2017. Data Management in Machine Learning: Challenges, Techniques, and Systems. In *SIGMOD*, Semih Salihoglu, Wenchao Zhou, Rada Chirkova, Jun Yang, and Dan Suciu (Eds.). ACM, 1717–1722.

[43] Arun Kumar, Jeffrey F. Naughton, Jignesh M. Patel, and Xiaojin Zhu. 2016. To Join or Not to Join?: Thinking Twice about Joins before Feature Selection. In *SIGMOD*, Fatma Özcan, Georgia Koutrika, and Sam Madden (Eds.). ACM, 19–34.

[44] Varun Kumar, Ashutosh Choudhary, and Eunah Cho. 2020. Data augmentation using pre-trained transformer models. *arXiv preprint arXiv:2003.02245* (2020).

[45] Dong-Hyun Lee. 2013. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on challenges in representation learning, ICML*, Vol. 3.

[46] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461* (2019).

[47] Guoliang Li, Yudian Zheng, Ju Fan, Jiannan Wang, and Reynold Cheng. 2017. Crowdsourced data management: Overview and challenges. In *SIGMOD*. 1711–1716.

[48] Yonggang Li, Guosheng Hu, Yongtao Wang, Timothy Hospedales, Neil M Robertson, and Yongxing Yang. 2020. DADA: Differentiable Automatic Data Augmentation. *arXiv preprint arXiv:2003.03780* (2020).

[49] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, AnHai Doan, and Wang-Chiew Tan. 2020. Deep entity matching with pre-trained language models. *PVLDB* 14, 1, 50–60.

[50] Hanwen Liang, Shifeng Zhang, Jiacheng Sun, Xingqiu He, Weiran Huang, Kechen Zhuang, and Zhenguo Li. 2019. Darts+: Improved differentiable architecture search with early stopping. *arXiv preprint arXiv:1909.06035* (2019).

[51] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. 2019. Fast autoaugment. In *NeurIPS*. 6665–6675.

[52] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable Architecture Search. In *ICLR*.

[53] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[54] Mohammad Mahdavi and Ziawasch Abedjan. 2020. Baran: Effective Error Correction via a Unified Context Representation and Transfer Learning. *PVLDB* 13,

11 (2020).
[55] Mohammad Mahdavi, Ziawasch Abedjan, Raul Castro Fernandez, Samuel Madden, Mourad Ouzzani, Michael Stonebraker, and Nan Tang. 2019. Raha: A configuration-free error detection system. In *SIGMOD*. 865–882.
[56] Andrew McCallum and Ben Wellner. 2005. Conditional models of identity uncertainty with application to noun coreference. In *NeurIPS*. 905–912.
[57] Venkata Vamsikrishna Meduri, Lucian Popa, Prithviraj Sen, and Mohamed Sarwat. 2020. A Comprehensive Benchmark Framework for Active Learning Methods in Entity Matching. In *SIGMOD*. 1133–1147.
[58] Zhengjie Miao, Yuliang Li, Xiaolan Wang, and Wang-Chiew Tan. 2020. Snippext: Semi-supervised opinion mining with augmented data. In *Proceedings of The Web Conference 2020*. 617–628.
[59] Tomas Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NeurIPS*, Christopher J. C. Burges, Léon Bottou, Zoubin Ghahramani, and Kilian Q. Weinberger (Eds.). 3111–3119.
[60] George A Miller. 1998. *WordNet: An electronic lexical database*. MIT press.
[61] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep learning for entity matching: A design space exploration. In *SIGMOD*. 19–34.
[62] Hao Nie, Xianpei Han, Ben He, Le Sun, Bo Chen, Wei Zhang, Suhui Wu, and Hao Kong. 2019. Deep sequence-to-sequence entity matching for heterogeneous entity resolution. In *CIKM*. 629–638.
[63] Tong Niu and Mohit Bansal. 2019. Automatically learning data augmentation policies for dialogue tasks. *arXiv preprint arXiv:1909.12868* (2019).
[64] Hyunjung Park and Jennifer Widom. 2014. CrowdFill: collecting structured data from the crowd. In *SIGMOD*, Curtis E. Dyreson, Feifei Li, and M. Tamer Özsu (Eds.). ACM, 577–588.
[65] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *NeurIPS*. 8026–8037.
[66] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. *OpenAI Blog* 1, 8 (2019), 9.
[67] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683* (2019).
[68] Erhard Rahm and Hong Hai Do. 2000. Data Cleaning: Problems and Current Approaches. *IEEE Data Eng. Bull.* 23, 4 (2000), 3–13.
[69] Alexander Ratner, Stephen H. Bach, Henry R. Ehrenberg, Jason Alan Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid Training Data Creation with Weak Supervision. *PVLDB* 11, 3 (2017), 269–282.
[70] Alexander J Ratner, Henry Ehrenberg, Zeshan Hussain, Jared Dunnmon, and Christopher Ré. 2017. Learning to compose domain-specific transformations for data augmentation. In *NeurIPS*. 3236–3246.
[71] Theodoros Rekatsinas, Xu Chu, Ihab F Ilyas, and Christopher Ré. 2017. HoloClean: Holistic Data Repairs with Probabilistic Inference. *PVLDB* 10, 11 (2017).
[72] Theodoros Rekatsinas, Manas Joglekar, Hector Garcia-Molina, Aditya G. Parameswaran, and Christopher Ré. [n.d.].
[73] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *NeurIPS*. 91–99.
[74] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).
[75] Sunita Sarawagi and Anuradha Bhamidipaty. 2002. Interactive deduplication using active learning. In *KDD*. 269–278.
[76] Jürgen Schmidhuber. 1987. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. Ph.D. Dissertation. Technische Universität München.
[77] Burr Settles. 2009. *Active learning literature survey*. Technical Report. University of Wisconsin-Madison Department of Computer Sciences.
[78] Vraj Shah, Arun Kumar, and Xiaojin Zhu. 2017. Are Key-Foreign Key Joins Safe to Avoid when Learning High-Capacity Classifiers? *PVLDB* 11, 3 (2017), 366–379.
[79] Kihyuk Sohn, David Berthelot, Chun-Liang Li, Zizhao Zhang, Nicholas Carlini, Ekin D Cubuk, Alex Kurakin, Han Zhang, and Colin Raffel. 2020. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *arXiv preprint arXiv:2001.07685* (2020).
[80] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *NeurIPS*. 3104–3112.
[81] Sheila Tejada, Craig A Knoblock, and Steven Minton. 2002. Learning domain-independent string transformation weights for high accuracy object identification. In *KDD*. 350–359.
[82] Paroma Varma and Christopher Ré. 2018. Snuba: Automating Weak Supervision to Label Training Data. *PVLDB* 12, 3 (2018), 223–236.
[83] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *NeurIPS*. 5998–6008.
[84] Ricardo Vilalta and Youssef Drissi. 2002. A perspective view and survey of meta-learning. *Artificial intelligence review* 18, 2 (2002), 77–95.
[85] Wei Wang, Meihui Zhang, Gang Chen, H. V. Jagadish, Beng Chin Ooi, and Kian-Lee Tan. 2016. Database Meets Deep Learning: Challenges and Opportunities. *SIGMOD Rec.* 45, 2 (2016), 17–22.
[86] Jason Wei and Kai Zou. 2019. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196* (2019).
[87] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3-4 (1992), 229–256.
[88] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *ArXiv* (2019), arXiv–1910.
[89] Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. 2019. Unsupervised data augmentation for consistency training. *arXiv preprint arXiv:1904.12848* (2019).
[90] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. 2018. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541* (2018).
[91] Dongxiang Zhang, Yuyang Nie, Sai Wu, Yanyan Shen, and Kian-Lee Tan. 2020. Multi-Context Attention for Entity Matching. In *Proceedings of The Web Conference 2020*. 2634–2640.
[92] Dan Zhang, Yoshihiko Suhara, Jinfeng Li, Madelon Hulsebos, Çagatay Demiralp, and Wang-Chiew Tan. 2020. Sato: Contextual Semantic Type Detection in Tables. *PVLDB* 13, 11 (2020), 1835–1848.
[93] Tianhao Zhang, Gregory Kahn, Sergey Levine, and Pieter Abbeel. 2016. Learning Deep Control Policies for Autonomous Aerial Vehicles with MPC-Guided Policy Search. *ICRA* (2016).
[94] Xiang Zhang, Junbo Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *NeurIPS*. 649–657.
[95] Chen Zhao and Yeye He. 2019. Auto-EM: End-to-end Fuzzy Entity-Matching using Pre-trained Deep Models and Transfer Learning. In *The World Wide Web Conference*. 2413–2424.