A Meta Learning-Based Approach for Zero-Shot Co-Training

GUY ZAKS AND GILAD KATZ
Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev, Be’er Sheva 8410501, Israel
Corresponding author: Gilad Katz (giladkz@bgu.ac.il)

ABSTRACT
The lack of labeled data is one of the main obstacles to the application of machine learning algorithms in a variety of domains. Semi-supervised learning, where additional samples are automatically labeled, is a common and cost-effective approach to address this challenge. A popular semi-supervised labeling approach is co-training, where two views of the data – achieved by the training of two learning models on different feature subsets – iteratively provide each other with additional newly-labeled samples. Despite being effective in many cases, existing co-training algorithms often suffer from low labeling accuracy and a heuristic sample-selection strategy that hurt their performance. We propose Co-training using Meta-learning (CoMet), a novel approach that addresses many of the shortcomings of existing co-training methods. Instead of employing a greedy labeling approach of individual samples, CoMet evaluates batches of samples and is thus able to select samples that complement each other. Additionally, our approach employs a meta-learning approach that enables it to leverage insights from previously-evaluated datasets and apply these insights to other datasets. Extensive evaluation on 35 datasets shows CoMet significantly outperforms other leading co-training approaches, particularly when the amount of available labeled data is very small. Moreover, our analysis shows that CoMet’s labeling accuracy and consistency of performance are also superior to those of existing approaches.

INDEX TERMS
Co-training, semi-supervised learning, meta-learning.

I. INTRODUCTION
The ease with which digital data can now be collected, processed and stored has paved the way for organizations to transform their operations through the use of machine learning (ML). In many cases, however, the lack of labeled data – not the overall lack of data – is the main barrier to the application of classification/regression algorithms in new domains. Data labeling is difficult because it often requires human involvement (i.e., tagging and labeling), which makes the process slow and expensive.

Both unsupervised and semi-supervised approaches have been proposed to overcome the lack of labeled data. Unsupervised approaches usually involve the use of clustering [51] or embedding [50], and they enable the grouping of similar items and/or the modeling of their contextual connections. While unsupervised methods are highly effective in domains such as Natural Language Processing, there is no guarantee that the data will be grouped in a way that facilitates the labeling process. Moreover, in cases where only a small amount of labeled data exists (which is the challenge addressed in this study), no clear way exists to propagate these labels to additional samples.

Semi-supervised learning combines a small set of labeled samples with a larger set of unlabeled samples in order to derive meaningful insights about the data. This ability to combine both types of data makes semi-supervised learning an attractive approach for the labeling of additional samples. Approaches of this type include the Expectation-Maximization (EM) algorithm [4], [8], the use of generative models [7], [21], and self-training [24], [46].

Another popular semi-supervised labeling approach is co-training, originally proposed in [1]. In this approach, the dataset – both labeled and unlabeled samples – is partitioned into two disjoint sets of features, and a separate learning model is trained on each (labeled) set. Each learning model then selects a small number of unlabeled samples which – according to its current understanding of the data – are highly likely to belong to a given class. These few samples are then assigned their supposed labels and added to the labeled set, and the learning models are re-trained on the augmented labeled set. This process is then repeated...
iteratively. Co-training is based on the intuition that because of the different features used to train each learning model, the two models have different perspectives on the data. These different perspectives lead to each model labeling samples the other would not, thus increasing the labeled set’s diversity and preventing over-fitting.

Despite significant improvements to the original co-training algorithm [30], [43], [55], the general approach of the algorithm remains similar: each learning model selected a small number of samples, with the selection being guided solely by the model’s confidence score. We consider this approach to have several shortcomings:

- **Greedy selection of single samples.** Although each co-training iteration adds a batch of new samples, each sample in this batch is selected greedily, without any consideration to the other samples. This could lead to non-complementary selection of samples and even potentially to the selection of opposing samples: near-identical samples assigned with opposite labels by different learning models.

- **No analysis of the learning models’ correlation.** One of the underlying assumptions of the co-training algorithm is that the learning models are not correlated since they are trained on disjoint feature sets. However, in the majority of datasets today (unlike in the original co-training study [1]) there is no clear and intuitive way of partitioning features. As a result, the partitioning is often done randomly. While it has been shown that co-training can still be effective in such cases [16], [55], current approaches offer no way of assessing the classification similarity of the participating learning models, nor do they have ways of adapting themselves to address it.

- **No consideration for dataset characteristics.** It is highly likely (based on studies such as [18]) that the efficacy of co-training algorithms will be affected by various traits of the dataset on which they are applied. Dataset characteristics such as size, number of features, and feature values distribution can all have an effect both on the performance of the co-training algorithm and on the characteristics of the most effective sample-selection strategy. To date, no co-training-based algorithm takes these factors into account.

- **No learning across multiple datasets.** Despite the fact that the labeled set of the analyzed dataset is often very small (which makes extracting useful insights difficult), no attempts have been made to leverage information from additional datasets in order to facilitate the co-training process (i.e., meta and/or transfer learning). In this study we propose Co-training using Meta-learning (CoMet), a meta learning-based approach for co-training. Our approach models the analyzed dataset – both labeled and unlabeled samples – and the performance of each learning model, and trains a meta-learning model to select the samples that will be added to the labeled set of samples. In addition to being able to dynamically adapt its sample-selection policy, CoMet also performs batch selection rather than the standard selection of individual samples in an attempt to ensure that the set of selected samples complement each other and the performance of the co-training algorithm. Additionally, this study is the first (to the best of our knowledge) to leverage information from previously-analyzed datasets in order to improve the co-training process.

Our contributions are as follows:

- We present CoMet, a meta-learning based approach for co-training. Our approach leverages meta-data both from the current dataset and from previously analyzed datasets to guide the sample-selection process.

- We empirically demonstrate the merits of our approach on a large group of datasets, achieving an improvement 42%-61% in error reduction compared to advanced, well-known and popular versions of the co-training algorithm.

- We show that CoMet’s sample-selection policy is highly accurate and results in a much lower percentage of mistaken labeling of new samples.

- We make our code publicly available here: https://github.com/ZaksG/MetaCoTrainingV2.

II. RELATED WORK

A. THE CO-TRAINING ALGORITHM

The original co-training algorithm [1] is presented in Algorithm II-A. Assume a dataset D with a set of features F of size N and M samples. The dataset consists of a small set of labeled samples L, a large set of unlabeled samples U, and a test set T. The following process is performed for k iterations:

1) Partition the set of features F into two subsets of features $F_1, F_2$ (line #2). This partitioning results in two disjoint “views” of L and U: $L_1, L_2, U_1$ and $U_2$ respectively (line #4).

2) Train two learning models $h_1$ and $h_2$ on $L_1$ and $L_2$ (lines #5-#6), and use them to classify $U_1$ and $U_2$ respectively (lines #7-#8).

3) For each possible class value (i.e., the feature in D denoting the classification), select a small fixed number of samples from each learning model – $h_1$ and $h_2$ – that received the highest classification confidence score (line #9).

4) Add these samples to labeled set $L$ with their (presumed) labels (line #10), and remove them from unlabeled set $U$ (line #11).

Upon the completion of this iterative process, we train the two final learning models on the augmented labeled set (line #13-#14). We then apply the models on the test set and combine their classification results to produce the final classification (line #15).

Co-training was originally applied to the problem of webpages classification. The authors divided the dataset into two mutually exclusive (and intuitive) views – page content and hyperlinks anchor text, assuming the two views were both sufficient and independent. Initially, co-training was mostly applied to datasets that complied with these assumptions of
The majority of the work in the field of co-training aims to improve the algorithm’s performance in this regard.

The Adaptive View Validation algorithm, proposed by Muslea et al. [29], attempts to determine whether a multi-view split of the data was sufficiently compatible with tasks such as co-training. The approach is based on meta-learning: the user supplies a dataset, two predefined views, and a list of tasks performed on those views. The algorithm then extracts meta-features based on the dataset and views (e.g., the minimum and maximum training error on each classifier, and the difference between these measures), and produces a score indicating the dataset’s split compatibility.

In Democratic Co-learning [54], no splitting of the data is used. Instead, three or more classifiers are trained on the entire training set, and majority voting is used to determine the labels. Another popular algorithm is Tri-training [55], which utilizes three learning models instead of the regular two. An unlabeled sample is only labeled by a given learning model $H_1$ if the two other models $H_2, H_3$ agree on the instance’s label. The labeling process continues until convergence, and the prediction defines by majority vote. Another version of this algorithm is Tri-training with disagreement [43], where a sample is labeled only where $H_2$ and $H_3$ are simultaneously in agreement among themselves and in a disagreement with $H_1$. The goal of this version of the algorithm is to encourage diversity in the labeled data.

In Self-training [30], the authors use a set of three classifiers, an unlabeled instance is labeled via majority rule. The final prediction is done using a single classifier. Rather than rely on voting or ensembles, Robust Co-training [44] applies canonical correlation analysis (CCA) to inspect the labels assigned to samples in an attempt to increase labeling accuracy. Self-paced Co-training (SPaCo) [25] proposes the use of sampling with replacement for unlabeled samples (i.e., newly-labeled items are replicated rather than removed from the unlabeled set). This approach enables the algorithm to self-correct its assigned labels at a later date. The authors of [16] propose combining the learning models trained across different iterations and using them as an ensemble. Given the fact that the learning models often change significantly due to the addition of new samples to a small labeled set, the resulting ensemble proved to be effective for text classification tasks.

### 2) LABELING EFFECTIVE SAMPLES

One of the underlying assumptions of the co-training algorithm is that the labeling of additional samples, i.e., the enlargement of the originally limited labeled set, will increase the effectiveness of the classifier. This assumption only holds, however, if the added samples are useful for the prediction process. Often enough, easy-to-label samples (the ones often favored for labeling by the co-training algorithm) will not improve the learning algorithm’s performance. Consider the following cases: if the newly-labeled samples are in regions that do not contribute to the model’s ability to analyze more “challenging” samples – either easy-to-classify outliers or

### B. ADVANCED CO-TRAINING ALGORITHMS

Subsequent work on the co-training algorithm mainly sought to improve two shortcomings of the algorithm. The first shortcoming is mistakes in the labeling process, which inject a great deal of noise into the resulting learning model. The second shortcoming is the labeling of ineffective samples, which refers to the selection of either “outlier” or too-easy-to-classify samples. Both types of samples contribute little to the classification process: the former is too different from other samples, while the latter is too similar to contribute any new information. We now review studies that aim to address these two challenges.

#### 1) IMPROVING LABELING ACCURACY

As shown in an analysis by [17], the co-training algorithm incorrectly labels on average approximately 30% of the samples it adds to the labeled set. These errors have a direct negative impact on the algorithm’s performance. As a result,
samples that are identical to previously-labeled samples – then the co-training process is unlikely to obtain useful new information.

When aiming to increase the efficacy of the labeled samples, previous studies have often sought to augment the co-training algorithm using active learning-based methods. Active learning [41] is a labeling approach in which a small number of samples are labeled by an oracle (e.g., a human expert) with perfect accuracy. Because such labeling is often expensive and/or time-consuming, active learning algorithms attempt to reduce the number of labeled samples by selecting the most informative ones – those that will best improve the classifier’s ability to distinguish between samples of different labels. Active learning is – in many ways – the mirror image of co-training: active learning is expensive but perfectly accurate, while co-training is inexpensive yet more error-prone.

Previous studies have proposed multiple ways to integrate co-training and active learning. In [56], the authors proposed a Gaussian random field model to integrate the two approaches. Corrected co-training [13], operating under the assumption that it is easier for a human expert to correct automatically-assigned labels, explored different levels of human involvement in the co-training process. Zhang et al. [53] proposed SSLCA, a co-training and active-learning method that labels the instances with the highest confidence while taking into account the labels of their nearest neighbors. Instances that are deemed informative are collected and sent to the active-learning component, which labels the data at the end of the learning process. An instance is then considered informative based on the differences the confidence level assigned to it and those assigned to its same-class neighbors.

In classic Co-training with two views and a binary classification task, it is possible for the two learning models to label the same sample with different labels. Muslea et al. [27] coined the term content point, which was used to describe such disagreement. The authors proposed the Co-Testing method, which operates similarly to the original co-training algorithm, but with the contention points being collected in a pool. An active-learner then queries the pool to label one random sample. The process is performed iteratively k times. The random selection of the pool could be improved through the use of more advanced strategies, although the authors chose to present the naive method due to its high performance. A further expansion of Co-Testing, which combines it with the Co-EM [30] algorithm, was proposed by Muslea et al. [28]. The authors proposed Co-EMT, whose first phase utilizes the probabilistic labeling of the Co-EM method [30]. The second phase of Co-EMT (applying the active-learner) utilizes the Co-Testing [12] approach to find the contention point.

It is important to note that all the studies reviewed here apply semi-supervised learning to tabular data. While we focus on the tabular domain as it is relevant to our proposed approach, semi-supervised learning is also frequently applied to other domains, most notably images [33, 37] and text [20, 52]. One must note, however, the fundamental differences between the tabular and the image/text domains. The latter domains have one significant advantage over the tabular: the ability to leverage pre-trained architectures (e.g., VGG [40] and Doc2Vec [23]). The high-dimensional nature of the image/text domains also often involve larger labeled sets than those used in tabular data. For example, [37] uses a labeled set of 30,000 samples in one of its experiments while co-training experiments on tabular data can be conducted on as little as 100 labeled samples.

C. SEMI-SUPERVISED LEARNING USING NEURAL NETWORKS

Another area of semi-supervised learning where meta-learning has been used to great effect is deep learning. While the “standard” co-training practice of training two separate classifiers (i.e., neural nets) on the labeled data has not, to the best of our knowledge, appeared in the literature, multiple studies in recent years have proposed other approaches. In [36], the authors propose an adaptive loss function that assigns weights to samples (both labeled and unlabeled) based on their importance. For the same setting, the authors of [19] analyze the entropy of the softmax function output in order to draw better decision boundaries between different classes. An approach for dealing with noisy labels was proposed by [6], who used two stacked softmax layers with the latter denoted as “noise separation layer” and used to model the noise of the original classification. The re-weighting of samples based on the “noisiness” of their labels has also received significant interest in recent years, with new neural architectures being proposed by studies such as [35, 42].

It is important to note that while the aforementioned methods proved effective, they differ from co-training in three important aspects. First, unlike co-training, these methods are only applicable to neural nets. This stems from the fact that all solution rely either on an adaptation of the neural net’s architecture or on the ability to analyze the gradients derived from the loss function. Co-training, in contrast, can be carried out with any type of learning algorithm. Secondly, the amount of labeled samples these methods require is considerably larger than those required by standard co-training methods. For example, the meta-learning in [42] is performed using a 100-neuron layer, while the smallest labeled training set in our experiments consists of only 100 samples. Secondly, these methods are mostly designed to operate on multi-label datasets (all reviewed studies are only evaluated on images) and it is unclear how they will perform on tabular (i.e., dense) binary datasets like those evaluated in this study. Finally, all reviewed studies are trained on the one dataset on which they are evaluated. CoMet, in contrast, is trained on multiple datasets and therefore requires no training when applied on new datasets while also enjoying a much larger training set.
D. UTILIZING META-LEARNING TO SUPPORT MACHINE LEARNING APPLICATIONS

Meta-learning, which is often described as the process of “learning about learning”, has been widely applied in a large variety of machine learning applications. By learning and representing the learning process itself, meta-learning can reduce running time, improve performance, and enable the efficient exploration of the problem space to discover superior solutions [11].

In recent years, meta-learning has been applied to multiple aspects of the machine learning process. Peng et al. [32] proposed various features for choosing the classification algorithm to be applied on a dataset. A similar approach was proposed by the recent work of [22], who used dataset and algorithm-based meta-features and embedding methods to recommend whole ML-pipelines (i.e., a sequence of algorithms). A somewhat similar meta-learning approach was proposed by [10], who used meta-learning combined with deep reinforcement learning to construct ML pipelines for previously-unseen datasets.

To the best of our knowledge, no studies exist regarding the application of meta-learning to analyze the inner-working or decision-making process of the co-training algorithm. We hypothesize that this is the case due to the fact that the small number of labeled samples prevent the creation of meaningful representations. In this work we propose a novel approach to enable the use of meta-learning in the above-mentioned context: instead of learning only from the current dataset, we train a meta-model on multiple previously-analyzed datasets. This approach has two significant advantages: first, it enables us to significantly increase the size of our meta-learner’s training set, thus increasing its effectiveness. Secondly, because we are able to train our meta-model “offline” (i.e., prior to the run on the new, previously-unseen datasets), our proposed approach is relatively computationally efficient.

III. PROBLEM FORMULATION

Assume a dataset $\mathcal{D}$ whose training set consists of a subset of labeled samples $\mathcal{L}$ and a subset of unlabeled samples $\mathcal{U}$, and whose test set is denoted as $\mathcal{T}$. The goal of the original co-training algorithm can be defined as

$$\arg \min_{\{u_i\} \in \mathcal{U}} \mathcal{E}(\mathcal{L}(\mathcal{F} \times (\mathcal{L} \cup \mathcal{U})), \mathcal{T})$$

where $\mathcal{E}$ is the loss evaluation function for the learning procedure $\mathcal{L}$, and $\mathcal{U}$ is a subset of $\mathcal{U}$ whose samples were iteratively assigned labels by the classifiers of the co-training algorithm in iterations $\{1..k\}$. Simply put, the goal of the co-training algorithm is to select samples from $\mathcal{U}$, assign them with labels, and add them to $\mathcal{L}$ so as to improve the performance of the learning procedure (i.e., the classifier).

While CoMet also seeks to minimize the loss function $\mathcal{E}$, two important differences exist. First, CoMet does not select individual samples but rather whole batches of samples. Secondly, the selection of the batches is aided by another learning procedure—the meta-learner. More formally, we define:

- Let $\mathbb{B}$ be the set of all candidate sample batches generated by CoMet from $\mathcal{U}$, where $\forall b_i, b_j \in \mathbb{B} [b_i = |b_j|]$.
- Let $m_{b_i}$, $m_d$ be the meta-features representing candidate batch $b_i$, and dataset $\mathbb{D}$ respectively.
- Let $\mathcal{M}$ be a meta-model (i.e., a learning algorithm) whose goal is to rank the candidate sample batches $\{b_1, \ldots, b_n\} \in \mathbb{B}$. We define the goal as

$$\arg \max_{b_i} \mathcal{M}(b_i, m_{b_i}, \mathbb{D}, m_d, \mathcal{L}, \mathcal{U})$$

We denote the batches chosen by $\mathcal{M}$ in iterations $\{1..k\}$ as the selected batches and denote them $\{b_1^{select}, \ldots, b_k^{select}\}$. The samples of these batches then make up $\mathcal{U}'$, and the goal of CoMet can be therefore defined as

$$\arg \min_{b_1^{select}, \ldots, b_k^{select} \in \mathcal{U}'} \mathcal{E}(\mathcal{L}(\mathcal{F} \times (\mathcal{L} \cup \mathcal{U}')), \mathcal{T})$$

IV. THE PROPOSED FRAMEWORK

A. OVERVIEW

Our proposed framework is presented in Figure 1. Our overarching goal is to create a co-training algorithm that (a) aims to select useful batches of samples rather than unrelated individual samples, and; (b) leverages meta-learning methods to make the batch-selection process dynamic and adaptable. To this end we employ four stages: meta-model creation (the “offline” phase), candidate batches generation, candidate batch ranking, and candidate batch selection (the three “online” phases).

In the meta-model creation phase we run multiple co-training experiments on a large set of datasets in order to ascertain what makes a batch effective. The chosen datasets have large variance in their characteristics—number of samples, number of features, feature type composition, etc.—designed to make the resulting meta-model as robust and generic as possible. For each dataset, we ran multiple co-training experiments, generating and evaluating the performance of multiple batches at every iteration of the co-training process. For each batch/dataset combination, we extracted multiple meta-features and paired them with the batch’s contribution to the performance on the dataset’s test set. These meta-features/performance pairs were then used as the training set for the training of our meta-model (in our experiments, we used the Random Forest algorithm to create the meta-model). This trained model was later used to identify promising sample batches in additional datasets. It is important to point out that this process was conducted “offline”, and therefore had no effect on CoMet’s response-time to new datasets (i.e., “online”).

The remaining three phases, are performed “online”, when CoMet is applied on previously-unseen datasets. The goal of the candidate batches generation phase is to generate a large and diverse set of candidate batches $\mathcal{C}^{cand}$ at every iteration $i$ of the co-training process. We select a fixed percentage of the top-ranked samples and generate multiple fixed-size sample
FIGURE 1. The meta-features extraction points during the co-training process. The co-training layer represents the original co-training algorithm, as explained in Section II-A. The meta-features extraction layer collects data regarding the dataset, the classifiers, the batches and their instances, and further explained in Section IV-D and Algorithm 2.

combinations (i.e., batches) that are evaluated in the next phases.

In the candidate batch ranking phase we extract the meta-features that enable our meta-models to produce a ranking for every batch $c_{i}^{cand} \in c_{i}^{cand}$. Finally, in the candidate batch selection we select a single batch $c_{i}^{select}$ and add its samples to the labeled set $L$. The selected batch replaces the individual samples chosen by the previous co-training algorithms, and the updates labeled set will form the ground truth for the next co-training iteration. CoMet’s online phase is presented in Algorithm 2. In the following subsections we describe the phases of the process in detail.

B. THE META-MODELS CREATION PHASE

The goal of this phase is to train a generic and robust meta-model for identifying effective (i.e., capable of improving the chosen performance metric) sample batches. To achieve this goal, we run multiple co-training experiments on a large set of diverse datasets, and generate multiple of batches in each iteration and analyze their performance.

The batch generation process is identical for the offline and online phases, and is presented in Section IV-C. The co-training process used in this phase closely follows the original co-training algorithm, including in the key point of using the top-ranked samples to create the batch that is added to the labeled set $L$ at the end of each training iteration. However, in addition to the above-mentioned batch, we also generate and evaluate additional 1,296 batches during each co-training iteration. These additional batches are generated from the 10% of the samples that received the highest confidence-scores. We used 10% as this number was found empirically to produce a diverse set of batches while preventing too high an error rate in the assignment of labels to individual samples.

For each of the generated batches, we then extract the relevant meta-features and assign it with one of two labels: “good” or “bad”. The label is assigned by adding the batch to the current labeled set, re-training the two classifiers and evaluating their contribution to the overall performance. If the addition of a given batch of samples sufficiently improves performance, the “good” label is assigned. The complete set of meta-features is presented in Section IV-D. It is important to note that the generated batches are not used to affect the progress of the co-training algorithm in any way, and are solely used for the task of meta-features generation.

Once the aforementioned process is completed, we will have created a large set of meta-features/label pairs. We are now able to train a supervised learning model, with the labels (“good”/“bad”) serving as the target feature and the meta-features serving as the features. The resulting learning model, to which we refer throughout this study as the meta-model, will be used at later phases of our approach to identify batches of samples from previously-unseen datasets that are likely to contribute to the CoMet’s performance. It is important to note that because its training set is extracted from a large group of diverse datasets (see Section V-B), the resulting meta-model is both generic and robust.

C. CANDIDATE BATCHES GENERATION

The goal of this phase is to create a diverse and effective set of candidate batches $C_{i}^{cand}$. When working towards this goal we need to balance our interest in generating a large number of candidates (which increases the chance of coming...
up with effective candidates) and the need to keep computational costs manageable. This phase is presented in line #8 of Algorithm 2.

We use the following steps to generate our set of candidate batches:

1) For each of our two classifiers \( H_1, H_2 \), we select the top 10% of individual samples assigned to each label. The samples are ranked based on the confidence scores assigned to them by the respective classifier/label combination. This step results in four sets of samples overall, two for each classifier.

2) For each classifier, we randomly select two samples per class from each of the two sets associated with it—four samples in total. We then combine the two sets of four chosen samples to create a batch of eight samples (equal to the number of samples chosen in the original co-training algorithm [1]).

3) We repeat the process to generate \( \binom{4}{2}^4 = 1296 \) batches for each co-training iteration. We reach this figure given that our datasets have two classes (i.e., binary datasets), four subsets of samples (i.e., class-partition combinations), and we chose to perform this process four times.

We use the top 10% of samples labeled by each classifier since we found it strikes the right balance between diversity and assurance in our experiments: on the one hand, our approach ensures that a large enough pool of samples exists to enable the creation of diverse batches. On the other hand, by choosing the top-ranked samples we have significant assurance that the labels assigned to samples of the batch (by the two classifiers) are indeed correct. Interestingly, our approach actually yields higher label accuracy than the original co-training algorithm which selects the top-ranked samples (i.e., those that have the highest classifier confidence). We elaborate on this point further in Section V-C.

### D. META-FEATURES EXTRACTION

The goal of this phase is to generate a set of meta-features for the generated candidate samples batches so that our meta-model can rank them based on their predicted effectiveness. To achieve this goal, we generate four types of meta-features: dataset-based meta-features, confidence-score distribution meta-features, batch-based meta-features, and instances-based meta-features. It is important to note that these meta-features are re-calculated at every iteration of the co-training algorithm, as a result of the changes in the labeled datasets and the classifiers. Next we elaborate on each group of meta-features.

1) **DATASET-BASED META-FEATURES**

The generation of these meta-features is presented in line #3 of Algorithm 2. The various characteristics of every dataset are likely to significantly affect the efficacy of the co-training algorithm in general, and the efficacy of each generated batch in particular. We therefore extract a large set of features designed to capture the essence of each analyzed dataset. We extract meta-features that model both simplistic traits (size, number of features, etc.) and more advanced ones (e.g., feature value correlations). It is important to note that some meta-features were calculated on the labeled set \( L \) while others were calculated on \( L \cup U \). Generally, we used the former setting when the label of the sample was needed. In cases where the label was irrelevant, we preferred the larger set size since it was more statistically significant. We generate three types of meta-features:

1) **General information**: general statistics on the analyzed dataset: number of instances and classes, number of features and their types, class imbalance, etc.

2) **Initial evaluation**: statistics on the performance of a classifier trained on the entire (non-partitioned) labeled dataset. The evaluation is conducted using 10-fold cross-validation, and the generated meta-features include descriptive statistics on the AUC, precision/recall values at various thresholds and log loss.

3) **Feature diversity**: we partition the dataset based on feature type—discrete or numeric—and conduct paired-t tests and chi-squared tests on every pair of features in each sub-group. We generate meta-features using the tests’ statistic values.

It is important to note that these meta-features are recalculated at the start of every co-training iteration (see line #2 in Algorithm 2).

#### Algorithm 2 The CoMet Algorithm “Online” Phase

**Input:** \( L \) - Labeled set, \( U \) - Unlabeled set, \( H \) - Classifiers set, \( D \) - Datasets set, \( (V_1, V_2) \) - views of dataset \( d \), \( M \) - the meta-model, \( T \) - the test set

**Output:** \( C \) – The final classification on \( T \)

1: \( MF_{ds} \leftarrow \emptyset \)
2: for \( d_i \in D \) do
3: \( MF_{ds} \leftarrow DatasetMetaF(d_i) \)
4: for \( k \) iterations do
5: \( q_1 \leftarrow TrainClassifier(H_1, L, V_1) \)
6: \( q_2 \leftarrow TrainClassifier(H_2, L, V_2) \)
7: \( MF_{ds} \leftarrow MF_{ds} \cup ScoreDistMetaF(q_1, q_2) \)
8: GenBatchCandidates \( \leftarrow \)
9: for \( batch \in BatchCandidates \) do
10: \( MF_{ds} \leftarrow MF_{ds} \cup BatchMetaF(batch) \)
11: for instance \( \in batch \) do
12: \( MF_{ds} \leftarrow MF_{ds} \cup InstMetaF(instance) \)
13: end for
14: end for
15: \( B \leftarrow selectBatchToLabel(MF_{ds}, M) \)
16: \( L \leftarrow L \cup B \)
17: \( U \leftarrow U \setminus B \)
18: end for
19: end for
20: \( C \leftarrow ProduceFinalClassification(L, V_1, V_2, T) \)
21: return \( C \)
Algorithm 2. This is necessary use to changes to the labeled and unlabeled sets (see lines #16-#17 in Algorithm 2).

2) CLASSIFIERS CONFIDENCE SCORE-BASED META-FEATURES

Both the original co-training algorithm and more recent approaches (see Section II) use heuristics to select the samples that are added to \( L \) each iteration. Some approaches [14], [26] seek consensus among their learning models, while other seek some level of disagreement [45], [48]. Rather than define a rule that is not likely to be optimal for all cases, we define a diverse set of meta-features designed to analyze the learning models’ score distributions — both individually and with respect to each other. These meta-features enable us to infer whether the learning models are effective and/or correlated in their performance. They are generated in line #7 of Algorithm 2, following the re-training of the learning models (lines #5-#6). The meta-features of this group can be partitioned into five groups:

1) **General statistics**: descriptive statistics of the confidence-score distribution per partition.

2) **Distribution type**: we perform goodness-of-fit tests to determine whether the confidence score distribution of each learning model fits a known distribution type—Gaussian, log-normal or uniform—by conducting the Shapiro-Wilk and Kolmogorov-Smirnov tests. We extract descriptive statistics on the tests’ p-value scores for each distribution type and use them as meta-features.

3) **Features correlation**: this set of features is calculated on the unlabeled set. For each learning model, we define a set of thresholds and partition the instances of the unlabeled set based on the labels assigned by the classifier. For example, if the threshold that was chosen is 0.7, all instances whose confidence score is \( \geq 0.7 \) will be assigned a label of 1 and the remaining samples will be assigned a label of 0. We then conduct statistical tests on the correlation of each feature across the partition (that is, whether the values distribution of the “label 0” samples correlate with those of “label 1”). We calculate T-statistic for numeric features and Chi-Square statistic for discrete features, and then derive descriptive statistics on the p-values.

4) **Comparison to previous iterations**: for each learning model, we compare its current confidence-score distribution on the unlabeled set to those of previous previous iterations. We compare the current iteration’s performance to those of 1, 3, 5 and 10 previous iterations. We extract two types of statistics: (a) descriptive statistics on the confidence score deltas, i.e., differences in confidence score values \( \Delta = (x_i - y_i) \) for each sample, where \( x_i \) and \( y_i \) are the current and previous values, respectively, and (b) paired T-test statistical tests on the score distributions.

5) **Temporal difference**: this set of meta-features was inspired by temporal-difference techniques [3], [47], which are often applied in the field of reinforcement learning. This set of meta-features was generated by: (a) adding the current candidate batch to the labeled set; (b) creating another version of the classifiers, trained on the new dataset, and; (c) re-calculating all the meta-features presented in this section. In essence, this set of meta-features provides a “sneak peek” into the future, thus enabling us to gauge the effect that the candidate batch will have on the learning models’ behaviour. It should be noted that while this set of features was computationally-heavy, it had significant contribution to CoMet’s performance.

3) **BATCH-BASED META-FEATURES**

We hypothesize that an effective batch of samples will be one whose instances support each other. By support we mean that the instances do not negate each other (e.g., near-identical instances with opposite labels) and whose feature composition is sufficient to affect the training of the next learning model. Moreover, we hypothesize that the samples of the batch should generally exist in the same “region” of the dataset, because this would enable the learning models
to obtain a sufficient amount of new information to improve their performance in that specific region. Based on these hypotheses, we create two groups of meta-features:

1) **Confidence-score distribution within the batch:** on the eight instances within the batch, we extract descriptive statistics and compare their confidence-score and feature-value distributions to previous iterations.

2) **Unlabeled set space:** we randomly sample up to 1,000 instances from the unlabeled set \( U \). We then generate a feature-centroid for the evaluated batch (i.e., an average of all feature values). For each dataset feature \( f \) represented in the centroid, we calculate the difference \( \Delta = C_f - B_f \), between the feature-centroid (denoted by \( C \)) and the batch (denoted by \( B \)). We then generate descriptive statistics of the various \( \Delta \) values. These meta-features are generated in line #10 of Algorithm 2, following the generation of all the candidate batches for this co-training iteration in line #8.

4) **INSTANCE-BASED META-FEATURES**

This group of meta-features is used to analyze each of the samples that make up a candidate batch. These meta-features are designed to enable the meta-model with information about each sample’s labeling classification consistency (by analyzing the scores it received from each learning model now and in previous co-training iterations) and analyze its similarity to each of the other samples that make up the batch. Simply put, this group of meta-features provides an in-depth look at each of the candidate batches. These meta-features can be divided into four groups:

1) **Instance percentile:** for each classifier, we calculate the instance’s confidence score-percentile among all other unlabeled set.

2) **Classifiers comparison:** the difference between the confidence scores for this sample among the two classifiers. In addition, we calculate the average confidence-score.

3) **Feature-based:** we compare the sample’s feature values to those of the other instances that make up the batch. We then extract descriptive statistics that serve as the meta-features.

4) **Previous iterations comparison:** Similarly to the temporal difference meta-features presented above, we extract the confidence for each sample the confidence scores assigned to it in previous co-training iterations and calculate the difference with the current iteration. We then derive descriptive statistics on this set of values.

These meta-features are generated in line #12 of Algorithm 2, following the extraction of the meta-features for the overall batch.

**E. CANDIDATES BATCH SELECTION**

The goal of this phase is to select a single batch and add it to the set of labeled samples \( L \). The selection is carried out using random forest algorithm. We select the top-ranked batch (see line #15 in Algorithm 2), with ties broken randomly, and then perform the same actions as the standard co-training algorithm presented in Section II-A, including the re-training of the two classifiers on the augmented labeled set.

In addition to the actions of the standard co-training algorithm, CoMet also requires that we recalculate our approach’s meta-features: the dataset-based meta-features are calculated following the update of the labeled and unlabeled sets, the classifier confidence score-based are updated following the re-training of the classifiers, and the batch and instance-based meta-features are updated following the generation of the new sample batches.

Upon the completion of all co-training iterations, the final classification of the test set is carried out in an identical manner to the original co-training algorithm. This part of the process is presented in lines #20-#21 of algorithm 2.

**Algorithm 3 Leave-One-Out Evaluation Framework**

**Input:** \( D \) - set of \( n \) datasets  

**Output:** \( Res \) - Evaluation Results set  

1: Create offline pre-trained model \( PT(d) \) for every \( d \in D \), using the combined meta-data of all other datasets \( D \setminus \{d\} \)

2: for \( d_i \in D \) do

3: for 10 seeds do

4: Split \( d \) to train and test sets by 70%-30% ratio

5: Split the labeled and unlabeled sets of \( d \) into 2 views \( (V_1, V_2) \)

6: for 20 iterations do

7: Select a batch of sample by using \( PT(d) \)

8: end for

9: \( Res_{d_i} \leftarrow CalculateResults(d_i) \)

10: \( Res \leftarrow Res \cup Res_{d_i} \)

11: end for

12: end for

13: return \( Res \)

**V. EVALUATION**

The goal of our evaluation is to determine whether our batch and meta learning-based approach outperforms leading co-training algorithms. Our evaluation includes two criteria: (a) overall performance – the final classification performance of the respective algorithms, and; (b) labeling accuracy – the “correctness” of the labels automatically assigned to new labels throughout the co-training process.

Our reason for including labeling accuracy in the evaluation is simple: the co-training algorithm is often wrong. As shown in [17], the co-training can assign the wrong label to a sample in over 30% of overall cases (our analysis later in this section supports these conclusions). While final performance is indeed important, we argue the process of obtaining the results is important both for reasons of explainability (the ability to convey the system’s rationale to users) and auditing.
A. THE EVALUATED ALGORITHMS

We compare CoMet to the following co-training algorithm:
- The original co-training [1], as explained in Section II-A.
- Tri-training [55]: Using a set of three learning models, an unlabeled instance is labeled by a given learning model \( H_1 \) only if the other two models \( H_2, H_3 \) are in agreement on the instance’s label. The labeling process continues until convergence, and the predictions are produced by majority voting.
- Tri-training with disagreement [43]: An extension of the Tri-training method. A given learning model \( H_1 \) can only label an instance if (a) the other two models \( H_2, H_3 \) are in agreement regarding its label, and; (b) \( H_1 \) disagrees with \( H_2, H_3 \). This approach is motivated by the goal of diversifying the training set and preventing the selection of “obvious” (and possibly duplicate or near-duplicate) instances.

These baselines were chosen both because of their popularity and high performance. It is important to note that while more recent variants of the co-training algorithm exist [20], [34], [39], [49], they are designed for the NLP or image-classification domains, and their adaptation to tabular datasets is not straightforward. Secondly, like our proposed approach, the chosen baselines do not require the analyzed data to have sufficient and independent views of the features, and are designed to operate on random partition of the features.

B. EXPERIMENTAL SETUP

We evaluate CoMet on 35 supervised binary classification datasets. Our chosen datasets are highly diverse in terms of size, class imbalance, number of attributes and composition of features (continuous and/or discrete). All datasets are available on the OpenML repository\(^1\) and their properties are presented in Table 1. It should also be noted that the number of evaluated dataset in this study in larger than in previous studies: [55], for example, conducted their evaluation on 12 datasets.

We used the following setting throughout the evaluation:
- All datasets were randomly partitioned into training and test sets, with 70% of the data assigned to the former. The training set was partitioned into a (small) labeled set and an unlabeled set.
- We evaluated five scenarios with different sizes of the labeled set. In the first scenario, the size of the labeled set was set to 100 samples. In the remaining scenarios, the size was set to 20%, 40%, 60% and 80% of the labeled set size. In all scenarios, all samples were randomly chosen from the training set (label ratios were maintained), with the remainder of the training set was used as the unlabeled set. The first set size was chosen because it was one of the set sizes used in the original co-training paper [1], while the other labeled set sizes were chosen because they were used in the two co-training baselines to which we compare CoMet.
- We used the random forest algorithm [31] as the learning algorithm of our meta-model and sorted its classification confidence score to rank the batches.
- The classification algorithm used by our two learning algorithms \( H_1, H_2 \) was logistic regression.
- CoMet selected eight samples at each iteration – four samples by each classifier – in accordance with the original co-training algorithms. The number of samples chosen by the baselines varied, and was determined by their reported settings.
- For each instance of the dataset used to train the meta-model, a batch was labeled as ‘positive’ if it improved the AUC metric by a value of at least 0.005. Otherwise, it was labeled as ‘negative’.
- For every dataset we ran 10 co-training experiments of 20 iterations each. The initial seed of labeled samples was randomly chosen for each experiment, and both CoMet and the standard co-training algorithm used the same seeds.
- We used the classification error as the evaluation metric. We calculate it using the formula

\[
\text{error rate} = 1 - \frac{\text{AUC}_f}{\sum L \cup C_{\text{select}}}
\]

where \( \text{AUC}_f \) is the AUC values obtained for \( L \cup C_{\text{select}} \), where \( C_{\text{select}} \) consists of all the samples added to the labeled set throughout the experiment.

Our method for evaluating CoMet’s performance is presented in Algorithm IV-E. We used a leave-one-out (LOO) approach, where all datasets except for the one currently being evaluated were used to train the meta-model (see line #1). In other words, since our evaluation set consisted of 35 datasets, for each dataset \( d_i \), we trained the meta-model using metadata remaining 34 datasets \( (d_j \in \mathbb{D} \text{ where } i \neq j) \). Following the creation of the meta-model, we carry out the remainder of the evaluation process described earlier in this Section (i.e., 10 runs per datasets, 20 co-training iterations per run). These steps are presented in lines #3-#10.

C. RESULTS AND DISCUSSION

The results of our experiments are summarised in Table 2. Additionally, a detailed per-dataset analysis presented in Figures 2 and 3 for the two most challenging setups of 100 and 20% labeled samples, accordingly. It is clear that CoMet outperforms the three baselines by wide margins: from an average error reduction of 61%-19% for the

---

\(1\)https://www.openml.org/
FIGURE 2. A comparison of CoMet and all participating baselines for each of the analyzed datasets when the size of the initial labeled set is 100 samples. The graph shows the final error rate achieved by each approach (lower is better), in addition to general information on each dataset: number of instances and number of features.

While it is clear that CoMet outperforms the baseline algorithms in all experimental setups (i.e., different labeled-set sizes), the differences in performance are especially large for the more challenging scenarios, namely the 100-sample and 20%-sample labeled sets, and become increasingly small as the percentage of labeled data increases. It is important to note that of our baselines, only the original co-training was originally evaluated on a labeled set size of 100 samples, while the smallest labeled set size used by the other baselines was 20%. We argue that the latter baselines rely on more complex models that naturally require larger training sets to perform effectively. CoMet’s offline meta-learning process, on the other hand, enables it to learn from tens of thousands of samples regardless of the labeled set size of the currently-analyzed dataset. This difference enables our approach to achieve superior performance. An additional analysis of the performance of the various algorithms, presented in Figure 4, further illustrates this point.

Given the significant impact the labeled set size has on performance, we wish to place the differences between the 100-samples and the 20% sets in perspective: while for smaller datasets (e.g., Cardiography with 2126 samples overall) the size differences between the two approaches are minor, for larger datasets (e.g., Mammography with 11,183 instances) the difference in labeled-set sizes may reach an order of magnitude. Moreover, the 20% setting results on average in a labeled set that consists of approximately 550 samples, 5.5 times the size of its smaller counterpart.

Finally, we compared CoMet to each of the baselines using the paired-t statistical test. The test was conducted using the individual runs on each dataset, giving us a population of 350 samples for each algorithm. The results of the analysis show that our approach significantly outperform all baseline for all labeled set sizes with statistical significance of \( p < 0.01 \). Additionally, we performed the same test on every possible pair of our baselines. We found no statistically significant difference between the tri-training and tri-training w/disagreement baselines. The original co-training was found to be significantly better than the other two baselines only for the 100-sample setting with \( p < 0.01 \) and for the 20% setting with \( p < 0.05 \). The weaker indication of significance stems mostly from the high volatility of the baselines, whose performance depends on a heuristic and partially random selection of samples which also leads to large differences in performance. In contrast, CoMet’s use of a larger training set, due to its offline training, enables it to achieve a more stable performance.

1) LABELING CORRECTNESS OF ADDED SAMPLES
As shown in [17], the effectiveness of the co-training process is damaged by the fact that a significant percentage (\( \geq 30\% \) on average) of added samples are incorrectly labeled. We hypothesized that one contributing factor to the CoMet’s superior performance was higher labeling accuracy, made possible by our use of meta-learning.
FIGURE 3. A comparison of CoMet and all participating baselines for each of the analyzed datasets when the size of the initial labeled set is 20% of the training set. The graph shows the final error rate achieved by each approach (lower is better), in addition to general information on each dataset: number of instances and number of features.

FIGURE 4. Distribution of the error rate over datasets and co-training methods. We show results for the two smallest (i.e., most challenging) initial labeled set sizes. It is clear that CoMet’s median error rate is the lowest of all the evaluated methods.

To test this hypothesis, we analyzed the average labeling accuracy of the two evaluated algorithms – CoMet and the standard co-training algorithm – across all 35 datasets (we compare CoMet to the original algorithm because they label the same number of samples at each iteration). The results of our analysis are presented in Figure 5, and they show that not only does our approach significantly outperform the baseline (by approximately 30%), but that CoMet was able to reach near-perfect labeling accuracy. Practically, CoMet correctly labels (on average) 2.05 more samples per iteration than the standard co-training approach (out of 8 samples). The results presented in Figure 5 are for the 100-sample labeled training set, but the results are the same for all initial labeled training set sizes.

FIGURE 5. The average percentage of correctly-labeled samples for CoMet and the standard co-training algorithm. The results are calculated separately for every training iteration and averaged over all the evaluated datasets.

The near-perfect labeling accuracy of our approach is interesting given the fact that we do not aim to optimize this metric directly. These results also raise an interesting question: if the labeling is so close to perfect, why do we report higher error rates (14%-19.5%) compared to the 1%-2% error of our labeling? Our analysis of the labeled samples reached the conclusion that CoMet’s meta-model enables it to identify “safe” samples for which there is a very small chance of misclassification. This ability makes the labeling more accurate, but it also limits CoMet’s ability to label “interesting” samples that offer better differentiation (e.g., samples from class A that are relatively close to samples from class B). As a result, we obtain multiple correctly-labeled samples and can safely increase the size of our labeled training set, but our ability to deal with difficult-to-classify samples is more limited.
This analysis leads us to conclude that a “bolder” labeling strategy that can better balance the risks and rewards of such an approach could yield better performance. One possible approach to achieving this is the use of deep reinforcement learning [5], and we consider this a promising venue for future work.

VI. CONCLUSION

In this study we present CoMet, a meta learning-based co-training algorithm. Our approach focuses on selecting batches of useful samples rather than individual samples, and uses meta-learning to model the possible impact of the analyzed batches on the training data and the learning models that take place in the co-training process. CoMet significantly outperforms the standard co-training algorithm, while also achieving near-perfect sample labeling accuracy.

In future work we intend to leverage our meta-learning approach to explore methods for large-scale sample labeling. Additionally, we will explore using our approach as part of a deep reinforcement learning mechanism that will enable us to more intelligently carry out sequences of actions rather than the step-by-step approach of the co-training algorithm.

REFERENCES

[1] A. Blum and T. Mitchell, “Combining labeled and unlabeled data with co-training,” in Proc. 11th Annu. Conf. Comput. Learn. Theory (COLT), 1998, pp. 92–100.

[2] J. Chan and I. J. Koprinska Poon, “Co-training on textual documents with a single natural feature set,” in Proc. ADCS, 2004, pp. 47–54.

[3] P. Cichosz, “Truncating temporal differences: On the efficient implementation of TD (lambda) for reinforcement learning,” J. Artif. Intell. Res., vol. 2, pp. 287–318, Sep. 1994.

[4] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the EM algorithm,” J. Roy. Stat. Soc.: B (Methodol.), vol. 39, 1, pp. 1–22, 1977.

[5] V. Francois-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, “An introduction to deep reinforcement learning,” 2018, arXiv:1811.12560. [Online]. Available: http://arxiv.org/abs/1811.12560

[6] J. Goldberger and E. Ben-Reuven, “Training deep neural-networks using a noise adaptation layer,” in Proc. 5th Int. Conf. Learn. Represent. (ICLR), 2017.

[7] Y. Heffetz, R. Vainshtein, G. Katz, and L. Rokach, “RankML: A meta learning-based approach for pre-ranking machine learning pipelines,” 2019, arXiv:1911.00108. [Online]. Available: http://arxiv.org/abs/1911.00108

[8] Q. Le and T. Mikolov, “Distributed representations of sentences and documents,” in Proc. 31st Int. Conf. Mach. Learn. (ICML), 2014, pp. 1188–1196.

[9] I. Liviерис, A. Kanavos, V. Tampakas, and P. Pintelas, “An auto-adjustable semi-supervised self-training algorithm,” Algorithms, vol. 11, no. 9, p. 139, Sep. 2018.

[10] F. Ma, D. Meng, X. Qie, Z. Li, and X. Dong, “Self-paced co-training,” in Proc. 34th Int. Conf. Mach. Learn. (ICML), 2017, pp. 2275–2284.

[11] G. Katz, C. Caragea, and A. Shabtai, “Vertical ensemble co-training for semi-supervised learning,” in Proc. 10th Int. Symp. Imprecise Probab.: Theories Appl., vol. 2, 2002, pp. 169–180.

[12] M. F. A. Hady, I. J. Koprinska Poon, and F. Schwenker, “Semi-supervised learning with noisy data based on robust optimisation,” in Proc. 11th Annu. Conf. Comput. Learn. Theory (COLT), 2000, pp. 621–626.

[13] N. Flam, A. Flach, C. Soares, and P. Brazdil, “Improved dataset characterisation for meta-learning,” in Proc. Int. Conf. Discovery Sci. Springer, 2002, pp. 142–152.

[14] C. Persello and L. Bruzzone, “Active and semisupervised learning for the classification of remote sensing images,” IEEE Trans. Geosci. Remote Sens., vol. 52, no. 11, pp. 6937–6956, Nov. 2014.

[15] M. Pal, “Random forest classifier for remote sensing classification,” Int. J. Remote Sens., vol. 26, no. 1, pp. 217–222, Jan. 2005.

[16] Y. Peng, A. P. Flach, C. Soares, and P. Brazdil, “An auto-adjustable semi-supervised self-training algorithm,” in Proc. 9th Int. Conf. Inf. Knowl. Manage. (CIKM), 2002, pp. 3–10.

[17] S. Qiao, W. Chen, Z. Zhang, B. Wang, and A. Yuille, “Deep co-training for semi-supervised image recognition,” in Proc. Eur. Conf. Comput. Vis. (ECCV), 2019, pp. 93–108.

[18] M. Ren, W. Zeng, B. Yang, and R. Urtasun, “Learning to weight examples for robust deep learning,” in Proc. Int. Conf. Mach. Learn., 2018, pp. 4334–4343.

[19] R. Mihalcea, “Co-training and self-training for word sense disambiguation,” in Proc. 8th Conf. Comput. Natural Lang. Learn. (CoNLL-HLT-NAACL), 2004, pp. 34–40.

[20] R. Mihalcea, “Co-training and self-training for word sense disambiguation,” in Proc. 8th Conf. Comput. Natural Lang. Learn. (CoNLL-HLT-NAACL), 2004, pp. 34–40.

[21] I. Livieris, A. Kanavos, V. Tampakas, and P. Pintelas, “An auto-adjustable semi-supervised self-training algorithm,” Algorithms, vol. 11, no. 9, p. 139, Sep. 2018.

[22] F. Ma, D. Meng, X. Qie, Z. Li, and X. Dong, “Self-paced co-training,” in Proc. 34th Int. Conf. Mach. Learn. (ICML), 2017, pp. 2275–2284.

[23] M. F. A. Hady, F. Schwenker, and G. Palm, “Semi-supervised learning,” in Proc. 10th Int. Symp. Imprecise Probab.: Theories Appl., 2017, pp. 169–180.

[24] C. Persello and L. Bruzzone, “Active and semisupervised learning for the classification of remote sensing images,” IEEE Trans. Geosci. Remote Sens., vol. 52, no. 11, pp. 6937–6956, Nov. 2014.
G. Zaks, G. Katz: Meta Learning-Based Approach for Zero-Shot Co-Training

[42] J. Shu, Q. Xie, L. Yi, Q. Zhao, S. Zhou, Z. Xu, and D. Meng, “Meta-Weight-net: Learning an explicit mapping for sample weighting,” 2019, arXiv:1902.07379. [Online]. Available: http://arxiv.org/abs/1902.07379

[43] A. Søgaard, “Simple semi-supervised training of part-of-speech taggers,” in Proc. Conf. Short Papers (ACL), 2010, pp. 205–208.

[44] S. Sun and F. Jin, “Robust co-training,” Int. J. Pattern Recognit. Artif. Intell., vol. 25, no. 7, pp. 1113–1126, Nov. 2011.

[45] J. Tanha, M. V. Someren, and H. Afsarmanesh, “Disagreement-based co-training,” in Proc. IEEE 23rd Int. Conf. Tools with Artif. Intell., Nov. 2011, pp. 803–810.

[46] J. Tanha, M. van Someren, and H. Afsarmanesh, “Semi-supervised self-training for decision tree classifiers,” Int. J. Mach. Learn. Cybern., vol. 8, no. 1, pp. 355–370, Feb. 2017.

[47] G. Tesauro, “Temporal difference learning and TD-Gammon,” Commun. ACM, vol. 38, no. 3, pp. 58–68, 1995.

[48] W. Wang and Z.-H. Zhou, “Theoretical foundation of co-training and disagreement-based algorithms,” 2017, arXiv:1708.04403. [Online]. Available: http://arxiv.org/abs/1708.04403

[49] J. Wu, L. Li, and W. Y. Wang, “Reinforced co-training,” in Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics: Hum. Lang. Technol., vol. 1, 2018, pp. 1252–1262.

[50] J. Xie, R. Girshick, and A. Farhadi, “Unsupervised deep embedding for clustering analysis,” in Proc. Int. Conf. Mach. Learn., 2016, pp. 478–487.

[51] R. Xu and D. Wunsch, “Survey of clustering algorithms,” IEEE Trans. Neural Netw., vol. 16, no. 3, pp. 645–678, May 2005.

[52] M.-L. Zhang and Z.-H. Zhou, “CoTrade: Confident co-training with data editing,” IEEE Trans. Syst., Man, Cybern., Part B (Cybernetics), vol. 41, no. 6, pp. 1612–1626, Dec. 2011.

[53] Y. Zhang, J. Wen, X. Wang, and Z. Jiang, “Semi-supervised learning combining co-training with active learning,” Expert Syst. Appl., vol. 41, no. 5, pp. 2372–2378, Apr. 2014.

[54] Y. Zhou and S. Goldman, “Democratic co-learning,” in Proc. 16th IEEE Int. Conf. Tools Artif. Intell., Nov. 2004, pp. 594–602.

[55] Z.-H. Zhou and M. Li, “Tri-training: Exploiting unlabeled data using three classifiers,” IEEE Trans. Knowl. Data Eng., vol. 17, no. 11, pp. 1529–1541, Nov. 2005.

[56] X. Zhu, J. Lafferty, and Z. Ghahramani, “Combining active learning and semi-supervised learning using Gaussian fields and harmonic functions,” in Proc. Workshop Continuum Labeled Unlabeled Data Mach. Learn. Data Mining (ICML), vol. 3, 2003, pp. 1–8.

GUY ZAKS received the B.Sc. and M.Sc. degrees from the Department of Software and Information Systems Engineering, Ben-Gurion University of the Negev, in 2019 and 2020, respectively. His thesis focused on various aspects of automated machine learning (AutoML). He is currently a Data Scientist at Syte.

GILAD KATZ received the Ph.D. degree from the Department of Software and Information Systems Engineering (SISE), Ben-Gurion University of the Negev, in 2015, the subject of his dissertation being the intersection between machine learning and text analysis. From 2015 to 2017, he was a Postdoctoral Fellow at Prof. Dawn Song’s Group, UC Berkeley. In 2017, he became a Faculty Member at the Department of SISE, Ben-Gurion University of the Negev. He has worked extensively in the areas of AutoML—the automation of machine learning processes and pipelines—and the utilization of deep reinforcement learning to this end. He is currently a Lecturer (an Associate Professor) with the Department of SISE, Ben-Gurion University of the Negev. He has also collaborated actively with other researchers in the cyber-security domain, particularly robustness, generative models, and malware detection. His research interests include deep learning and deep reinforcement learning.