Picking groups instead of samples: A close look at Static Pool-based Meta-Active Learning

Ignasi Mas
Universitat Politècnica de Catalunya
Barcelona
i.masmend@gmail.com

Josep Ramon Morros
Universitat Politècnica de Catalunya
Barcelona
ramon.morros@upc.edu

Verónica Vilaplana
Universitat Politècnica de Catalunya
Barcelona
veronica.vilaplana@upc.edu

Abstract

Active Learning techniques are used to tackle learning problems where obtaining training labels is costly. In this work we use Meta-Active Learning to learn to select a subset of samples from a pool of unsupervised input for further annotation. This scenario is called Static Pool-based Meta-Active Learning. We propose to extend existing approaches by performing the selection in a manner that, unlike previous works, can handle the selection of each sample based on the whole selected subset.

1. Introduction

In a standard supervised classification problem, the system learns from several labeled samples from each class during the training stage, in order to classify new samples at test time. However, annotated data is not always available. Active Learning (AL) suggests a scenario where a specific cost is associated to labeling. This is a common setting in cases such as a company that has to pay some employees for labeling data.

On the other hand, Few-Shot Learning [31] suggests another scenario where the system is limited to train over a few samples per class. The extreme case is One-Shot Learning, where only one sample per class is available for training. There are several ways to approach the Few-Shot Learning task. Methods include getting more data (*e.g.* multimodal learning [3], domain adaptation [30] or data augmentation, depending on the available resources) or learning from data on similar problems.

Each Few-Shot problem can be solved using classical supervised learning: given a set of labeled data (the problem training set), the model is trained and it is expected to generalize on new samples. Later, the performance is evaluated with different samples of the same problem (its test set). We refer to this process as the Learning level.

We may want a system capable to adapt to a variety of problems for which it has not been specifically trained. One solution is to present the system several Few-Shot problems, different from the ones we will solve at test time but of similar nature. For each of these problems, annotated data should be available. We can train such a system by asking it to solve each one of these problems, analyze the results and update the model so that the error is minimized. This way, the system can generalize to new problems as it learns how to learn each problem. This process is named Meta-Learning because it is performed above the scope of a particular problem. At this level, there is a meta-training step (the process described before to adjust the model) and a meta-test step, to validate the complete system. The meta-test step consists of applying the system obtained in the meta-training step to a different set of problems (the meta-test set) and evaluating the results. The meta-test evaluation is the aggregation of all the single problem evaluations. We call the process where this pipeline works the Meta-Learning level.

The cross-over of Meta-Learning and Active Learning leads to Meta-Active Learning, which suggests the use of Meta-Learning to solve Active Learning problems at the Learning level. The AL component that performs the selection has to be trained as well. Clearly, the learning process involved here can not be performed at the Learning level (within the scope of a single problem) because the goal is to apply the automatic selection to new problems where no labeled data is available. Instead, a Meta-Leaning approach described in the above paragraph can be used to teach the AL component how to select the optimal samples to label.

The scenario where the system selects the samples from a pool of unlabelled data is called Pool-based Active Learning, in contrast with the case where samples arrive sequentially without any clue about future samples, that is named Stream-based Active Learning. One common problem with Pool-based Active Learning is that in the real world the system cannot always ask feedback without restrictions from the human annotation agent (called Oracle). This problem can be alleviated by using the so called Batch mode, where a batch of samples is selected before asking for labels, so that

the selection does not require feedback from the Oracle for each sample. However, it still relies on a process where the Oracle is asked several times sequentially. Instead, we will focus on Static Pool-based Meta-Active Learning, where the system selects the whole subset of samples before asking for their labels. Thus, for each problem the Oracle is asked just once. More insight into the different scenarios is given in 2.2.

The task of Static Pool-based Meta-Active Learning has already been studied by Contardo *et al*. in [6], where each sample is scored with a probability of being selected. Their approach allows learning a selection strategy that favors useful and representative data over deviant examples in a single step. However, it has some limitations which, depending on the scenario conditions, can be very harmful. We focus on the issue that each sample is selected independently (sample-focused), and in further sections we explain why it is an issue.

In our work, we propose a redefinition of the selection strategy to make it group-focused as opposed to sample-focused. This means that it should select each sample depending on the rest of the selected samples. This consideration should give the capacity to handle a final subsampling of the training data in a way that considers all possible groups of selection, so the probability is estimated per group instead of per sample.

We study the improvement of this method over the single step estimation. Furthermore, we provide results on the Omniglot dataset [15] and reason about the performance of our approach on a computer vision challenge.

2. Related work

2.1. Meta-Learning

Meta-Learning has been studied from many different approaches. Santoro *et al*. [21] proposed to solve the One-Shot Learning problem from a Memory perspective, following the idea presented by Graves *et al*. [11]. Later, the idea of metric learning introduced a new family of algorithms for Meta-Learning [29, 23, 14, 24]. Another approach suggested to find a proper parameter initialization to generalize enough for all the problems, proposed by Finn *et al*. [7] and extended by several works [17, 8, 33]. The last family of methods follows the idea of learning a proper optimizer [1, 18, 16].

2.2. Active Learning

All Active Learning problems share the property of having a cost assigned to the labels. With that in mind, we can define two scenarios.

The first one (Stream-based Active Learning) corresponds to the case where an agent receives data in an online manner (i.e. samples arrive consecutively and there is no clue about future samples) and has to decide either to label it or not. Many approaches have been proposed for solving this case [9, 20].

The second scenario (Pool-based Active Learning) gives the agent access to all the (unsupervised) data at once. Depending on the specific conditions of the problem different subtypes of Pool-based Active Learning can be defined. On a Static scenario, the selection is made at once before asking the labels to the Oracle. On a Sequential scenario, the system gets feedback from the Oracle for part of the selected subset before continuing the selection. This difference is crucial since the second scenario is not always feasible, as stated in [12]. Few Static Pool-based solutions have been proposed yet [34].

In Sequential Pool-based Active Learning, two modes can be defined: Single-instance mode [5] or Batch mode [13, 10]. In the first one each step is performed over a single sample while in the second one it is applied over a batch.

2.3. Meta-Active Learning

Meta-Active Learning still has few proposals. Most of them make use of Reinforcement Learning to guide the learning across Learning problems by exploring the performance when each given subset is selected. There are works for both the Stream-based scenario [32] and the Pool-based Meta-Active one in all its settings (specified in 1): Sequential Single-instance mode [2], Sequential Batch mode [19] and finally the Static scenario (where the subset is selected before labeling it). This last one was studied by Contardo *et al*. [6], and it is the focus of our work.

3. Method

3.1. Scenario

From now on we will differentiate between two levels, named Meta-Learning and Learning. Each Active Learning problem is solved at the Learning level (it learns from its available data), while the pipeline which updates the model is driven in the Meta-Learning level (through different Active Learning problems).

The detailed process of a single problem at the Learning level is illustrated in Figure 1, which uses as an example a binary classification problem between monkeys and fishes.

A generic problem p_i is specified as follows:

- K classes to classify.
- N samples belonging to any of the K classes (N samples in total, without restrictions on how they are distributed along classes), which build the labeled training set S_{train} and its unlabeled version U_{train}.
- M samples belonging to any of the K classes, that build the labeled set S_{test}, which we split between its unlabeled set U_{test} and its labels Y_{test} (for evaluation).

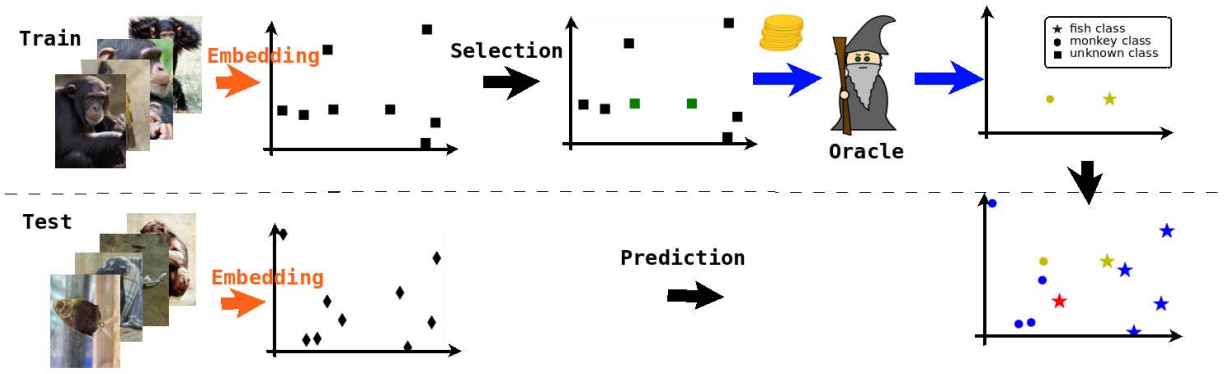


Figure 1: Example of a problem at the Learning level (binary classification between monkeys and fishes). Data samples are split into two sets, training and test, and embedded into an F-feature space ($F = 2$). The budget allows picking two samples ($B = 2$) from the training set, which are sent to an Oracle that returns their true labels, creating the supervised training set. A prediction algorithm (1-NN classifier) is applied to the test set based on the supervised training set. Most of the predictions result in hits, in blue, and a single fail, in red.

- A given budget B that determines the number of samples that we will be able to label from the unlabeled training set U_{train}, to get a labeled subset D.

As Figure 1 shows, to solve an Active Learning problem the process below is followed:

1. Select B samples from the represented U_{train} set and send them to the Oracle who should return the labeled subset D.
2. Predict classes for all samples in U_{test} based on D, giving $\hat{Y}_{\text{test}}$.
3. Evaluate $\hat{Y}_{\text{test}}$ against Y_{test} (true labels).

The pipeline at the Meta-Learning level is depicted in Figure 2. In order to build this pipeline, we need a set of classes C, where for each class a pool of samples and its associated labels are available. Ideally, at a high level, the classes should be uniformly distributed according to what is expected at production (*e.g.* if we expect to work on problems of classification of animals in general, classes for different animals should be used, trying to cover all the diversity of animals, instead of classes of animals and classes of cars).

The idea is to build a pipeline where several Learning problems guide the (meta-)training of the algorithm and allow a (meta-)evaluation of it. The process is driven through the following steps:

1. Split classes in C into $C^{metatrain}$ and $C^{metatest}$.
2. Generate two (meta-)sets of problems for both meta-training and meta-test stages, $P^{\text{metatrain}}$ and P^{metatest}.
3. Generate each problem p_i in the following way:
   (a) Select K classes from either $C^{\text{metatrain}}$ or C^{metatest}.
   (b) Generate train and test sets S_{train} and S_{test} by picking N and M samples and their respective labels from the K classes. Furthermore, create their unlabeled versions U_{train} and U_{test} (by ignoring their labels). Preserve the test labels Y_{test}.
   (c) Specify the budget B and number of classes K.

This setting allows us to (meta-)train by, at each epoch, randomly selecting a problem from $P^{\text{metatrain}}$, using the model to solve it and evaluating results. Those results are then used to update the model before the new epoch.

At (meta-)test time the process is repeated using the problems in P_{metatest} set to evaluate the results. This time, the model is not updated.

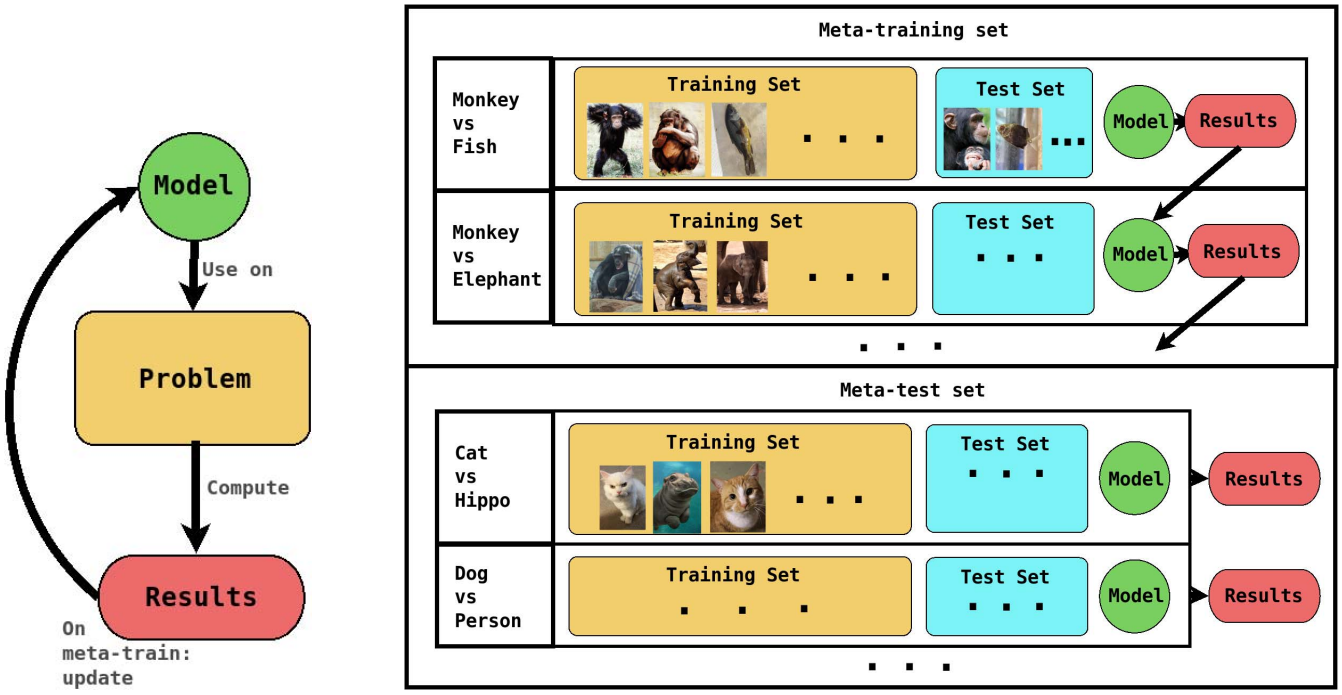
Note that we have not specified any model or algorithm for any step on the learning problem, we even ignored which kind of update is applied to the model or which results are used to do it. That is because the scope of this section is just to state the scenario to work on.

3.2. Sample-focused method

In [6] the model is split in 3 parts: Representation Model (RM), Selection Model (SM) and Prediction Model (PM).

The Representation Model finds a common F-feature space to embed samples for both SM and PM, and it is learned through the Meta-training (meta-)stage alongside other components.

The Selection Model gets U^R_{train} and should return D from the Oracle. Actually, [6] suggests a slight variation of that, where the SM returns a probability distribution α for selecting the samples in U^R_{train} as a Multinomial distribution where $\sum_{i=1}^{N} \alpha_i = 1$. Furthermore, in the Meta-training



(a) Iterative pipeline

(b) Unrolled version of the pipeline

Figure 2: Pipeline followed to guide the Meta-Learning process. At each epoch, a problem like the one represented in Figure 1 is presented and the model updates based on the results it gets. Note that at each epoch the presented problem has, for both the training and test sets, samples from a different combination of classes, while within the problem both sets have different samples from the same combination of classes. Furthermore, there is no class overlapping between the classes in the Meta-training and Meta-test (meta-)sets.

(meta-)stage the SM also returns the possible samplings D_α given a budget B. On the other hand, on the Meta-test (meta-)stage it just returns the B samples with highest probability in α, getting $D_{\alpha_{\text{MAX}}}$. In [6] the SM is implemented using a bidirectional GRU [4] since the input is considered a sequence of samples. We will discuss about the convenience of this choice in the following sections. This SM is learned alongside the RM, through the Learning problems in the Meta-training (meta-)stage. The group of all possible subsamplings can be represented as $D_\alpha = \{D_\alpha^1, D_\alpha^2, ...\}$ where each D_α^j is a single subsampling.

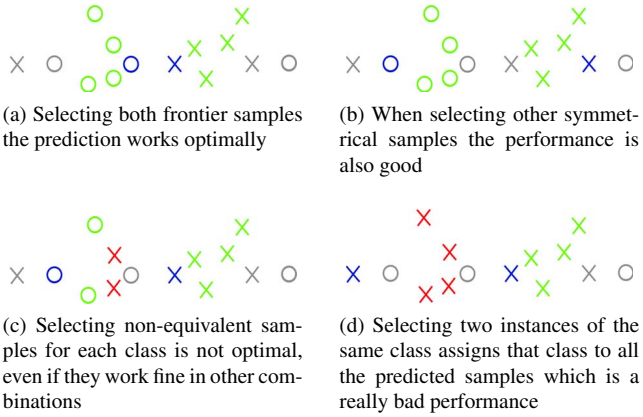
The Prediction Model gets both the represented unlabeled test set U_{test}^R and a labeled set D_α^j and predicts the classes of the test set $\hat{Y}_{test}^j$. More concretely, the classification of a test sample is performed by assigning the class with the maximal sum of metric similarities between the test sample and the train samples belonging to this class.

Since this PM does not require learning, only the RM and SM are learned, and they are trained jointly during the Meta-training (meta-)stage. The loss which updates the model is a Policy Gradient [26] that explores all the sampling space D_α from the distribution α and their rewards (based on the error of their prediction). The use of Policy Gradient to evaluate the whole probability distribution is justified because single samplings are not differentiable and therefore we cannot backpropagate from them. The loss is computed as follows:

$$\mathcal{L} = \sum_j -\text{log\_prob}(D_\alpha^j) r^j \tag{1}$$

where r is the reward of a given sampling computed as $r = -d(\hat{Y}_{\text{test}}, Y_{\text{test}})$ and $d(\hat{Y}_{\text{test}}, Y_{\text{test}})$ is the error between the prediction $\hat{Y}_{\text{test}}$ over the true labels Y_{test}. On the other hand, $\text{log\_prob}(D_\alpha^j) = \log(\text{prob}(D_\alpha^j))$ and the probability distribution vector α can determine the probability of a given sampling, i.e. $\text{prob}(D_\alpha^j)$. This $\text{prob}(D_\alpha^j)$ can be computed from the probability of the individual samples defined as α_i. For example, for a budget $B = 2$, the probability of a sampling of the first and second sample may be $(\alpha_1 \alpha_2)$.

This loss only gets high values on high rewards with low probabilities, so it tends to make the most profit on high rewards by penalizing the behavior that gives them a low probability.



(a) Selecting both frontier samples the prediction works optimally

(b) When selecting other symmetrical samples the performance is also good

(c) Selecting non-equivalent samples for each class is not optimal, even if they work fine in other combinations

(d) Selecting two instances of the same class assigns that class to all the predicted samples which is a really bad performance

Figure 3: Example of the effect of each kind of selection using a binary classification problem ($K = 2$, illustrated as crosses and circles) with a budget $B = 2$. In grey and blue there are the training samples, being blue the selected ones, while green and red are predicted test samples, green for hits and red for misses.

3.3. Limitation of Sample-focused method

The previous method is useful to avoid getting undesired samples, such as outliers, that will result in a high error.

However, it only uses the probability of selecting each individual sample *independently of the rest of the selected samples*. Nevertheless, a single sample can work very differently depending on the rest of the selected samples, as illustrated in Figure 3. There is one clear case of these limitations, which is when all the selected samples are from the same class. Even if the samples are very representative of their class, the result will obviously be very bad since all the test samples will be predicted as the same class.

3.4. Proposed methods

3.4.1 General architecture

We defined a method following the same pipeline as in 3.2. However, we extended it to overcome the issues mentioned in 3.3. Unlike the Sample-focused method, which just defines a single probability distribution for samples, we propose two ways of handling combinations of samples: the Iterative method and the Combinatorial method.

- The **Iterative method** consists of picking the samples one by one until the budget B is met. Moreover, we aggregate to the SM input the information about which samples have already been selected. The idea is to make the selector sensible to the already selected samples at each moment.
- The **Combinatorial method** consists of computing a single probability distribution (as in the method in 3.2), but unlike the Sample-focused method, this distribution is computed over combinations of samples. The length of the computed probability distribution vector α increases exponentially with the budget B.

3.4.2 Representation model

We used the convolutional part of VGG-16 [22], pretrained on ImageNet, and we stacked a Fully Connected layer to the desired F-feature space. We used $F = 128$.

3.4.3 Selection model

Our main improvements are focused on this component. The goal is to obtain, from an input vector of N samples (vectors of size F), the probability distribution vector α.

For the Iterative method, we append to each input sample feature vector an additional binary feature indicating whether it has been already selected or not, thus getting a $(F + 1)$-feature vector for each sample. We propose to scale each embedded feature on the range $[-1, 1]$ and the new binary feature as $\{0, F\}$, getting a vector of N ($F + 1$-dimensional) samples.

The Combinatorial method receives U^R_{train} and then represents all possible combinations by just stacking the corresponding B samples for that combination, getting a vector of N^B ((BF)-dimensional) samples. From this vector it needs to return α as a vector of size N^B, being the probabilities of each combination.

As for the component, we need a model that takes a number of input samples (N vectors of size $F + 1$ for the Iterative method and N^B vectors of size BF) and returns a vector α with the same size as the input, where each element refers to each input element (sample or combination) but taking into account the rest of elements in a sequence to sequence manner [25]. The model selected for this task is a *Bidirectional GRU* with a stacked Fully Connected layer at the end for the resulting features (as in [6]). We also use Softmax as the activation function. Additionally, the probability of selecting the samples already selected is set to 0.

There is something more we should take into account when generating α. The input U^R_{test} is actually a set, not a sequence, which means that its samples are ordered randomly and for different problems there is no relationship between their orders. However, Recurrent Neural Networks are designed to handle sequences. To overcome that issue we propose one of the solutions that Vinyals *et al*. suggest in [28], which is to force an artificial order. We propose to order the samples according to their distance to a reference. Among different distances and references that have been tested, the one that has given us the best results for the Iterative method is the Euclidean distance to the centroid for the already selected samples. For the Combinatorial method, ordering to the centroid of the already selected

samples is not possible (since selection is done in a single step) so we considered ordering by Euclidean distance to the U^R_{test} centroid.

3.4.4 Prediction model

We have based the PM on the distances between training and test vectors (in our case, Euclidean distance) as in [6]. More concretely, we have computed the probability to belong to each class as follows:

$$\hat{y}^{c_j}_{test_i} = \frac{\sum_{x_{\text{train}_k} \in D_\alpha, y_{\text{train}_k} = c_j} \text{dist}(x_{\text{train}_k}, x_{\text{test}_i})^{-1}}{\sum_l \sum_{x_{\text{train}_k} \in D_\alpha, y_{\text{train}_k} = c_l} \text{dist}(x_{\text{train}_k}, x_{\text{test}_i})^{-1}} \quad (2)$$

3.4.5 Loss

Policy Gradient requires the exploration of all possible samplings at (meta-)training time. In the Iterative method, the probability distribution is computed for each selected sample in an iterative way (given the previously selected samples). For this reason, if all samplings are explored at each selected sample, a probability distribution needs to be computed for each possible previous sampling. The probability of a final sampling is the product of the probabilities of the selected sample in each selection step in B.

We propose to compute the reward as a value that decreases as the error increases in an exponential way, where $r = e^{-d(\hat{Y}_{\text{test}}, Y_{\text{test}})}$. We use Cross Entropy as the error metric since it is a typical classification loss (so $d(\hat{Y}_{\text{test}}, Y_{\text{test}})$ is the Cross-Entropy error between $\hat{Y}_{\text{test}}$ and Y_{test}).

3.4.6 Training procedure

The procedure to guide the Meta-training consists of computing the loss for each epoch and updating the model with it. Note that the loss does not depend just on the quality of the model but also on the difficulty of the specific random problem, which can undesirably guide the update. To overcome this issue we propose to use several problems per epoch to smooth the randomness of the epoch. We use 50 problems per epoch. We just need to sum or average the losses of all the problems.

As already mentioned, the model is composed of RM, SM, and PM. However, PM is not trainable, so the optimizer will actually update RM and SM. We propose to pretrain RM for classification problems (through plain Meta-Learning) and then freeze it and just train the SM in the Meta-Active Learning setting.

4. Experimental validation

4.1. Experiments setting

Omniglot [15] is a dataset of handwritten characters specifically created for Few-Shot Learning. It consists of 50 alphabets.

Each alphabet is a group of characters, which are the actual classes. We split the alphabets into three disjoint (meta-)sets: (meta-)training, (meta-)validation and (meta-)testing alphabets. The method described in 3.1 is used: we construct several Learning problems to guide the (meta-)training of the algorithm and allow a (meta-)evaluation. For each (meta-)set, problems are generated by picking one random alphabet and selecting K random classes from it. The problems consist in classifying samples from these K classes. For each problem p^s_i, we create the training and test sets with samples from the selected classes.

The key is that any alphabet will not be found in more than one (meta-)set, but the problems will be of similar nature: classification problems of K characters on the same alphabet, but with a random alphabet on each problem.

Using 30 (meta-)training, 10 (meta-)validation and 10 (meta-)test alphabets, we defined a total of 4000 (meta-)training, 500 (meta-)validation and 500 (meta-)test problems of $K = 2$ and $B = 2$. That means that we are actually building a Meta-Active Learning setting where we want to learn to convert Active Learning problems to Binary One-Shot Learning problems and solve them through Meta-Learning.

Furthermore, we define $N = 15$ (15 unlabeled training samples from which we want to label 2 of them) and $M = 30$ (30 test samples to predict).

With that setting we compute two metrics: Multi-class ratio, which tells how many problems (relatively) have been solved selecting samples from different classes (a metric that makes sense for binary One-Shot Learning classification problems, i.e. $K = 2$, $B = 2$) and final accuracy.

4.2. Results

For fair comparison, we have trained all the approaches on the same benchmark (meta-)training set and evaluated on also the same (meta-)testing set, where the set of (meta-)training and (meta-)testing (and also meta-validation) problems fulfill that $P_{\text{train}} \cap P_{\text{test}} = 0$. The results obtained for the different approaches on the evaluation over the (meta-)testing set are presented in table 1.

First, as a worst-case scenario, we have tested a system where the Active Learning component selects the samples randomly (row 1). We have also made one experiment evaluating all possible selections (over the samples represented with the pretrained RM) and keeping the best one. The rest of the experiments (corresponding to the proposed approaches) are on this list:

		Multi-class ratio	Mean accuracy
1	**Random selection**	0.246	0.5385
2	**Sample-focused method (previous)**	0.416	0.4981
3	**Iterative method (unordered)**	0.278	0.5373
4	**Iterative method (ordered)**	0.784	0.6473
5	**Combinatorial method (ordered)**	0.374	0.5615
6	**Best selection**	0.998	0.8675

Table 1: Results for each approach, explained in 4.2

- **Sample-focused method:** We have replicated the method defined by [6].

  The results are presented in row 2. Comparing with the Random Selection (row 1), we see that there is no increase in final accuracy even having a higher Multi-class ratio, which shows that even if we pick samples from different classes we can have bad results (*e.g.* example *c* in Figure 3).

- **Iterative method, unordered samples:** In this experiment, we evaluate the Iterative method described in 3.4, without sample order.

  The results are presented in row 3. There is a slight improvement in accuracy. Something also noticeable is the decrease of the Multi-class ratio. This could be caused simply because, without order, the algorithm still does not find a proper behavior.

- **Iterative method, ordered samples:** At this point, we study the effects of imposing an order to the input of the SM as described in 3.4. As already told, this order consists of Euclidean distance to the previously selected samples centroid.

  The results are presented in row 4. Here we note a substantial improvement on both Multi-class ratio and on final accuracy. That is because the order helps the SM to better handle the selection depending on how far the samples are from the already selected ones. This ordering gets samples that tend to be from different classes (although not always, it is a difficult unsupervised task).

- **Combinatorial method:** Finally, we have experimented with the method explained in 3.4. In this case, we force the order at the input as the Euclidean distance to the pool centroid (there is no other possible origin for this method).

  The results presented in row 5 show a decrease for both the Multi-class and for the final accuracy with respect to the Iterative method with order. However, the reason may be that the order considers the pool centroid as the origin. This ordering has also been used with the Iterative method, with poor results.

5. Conclusions

In this work, we face the problem of Static Pool-based Active Learning as a Meta-Active Learning problem. We present two main contributions, and we improve the previously defined approaches within the same scenario.

The first one is making the selection Group-focused (within the Static scenario), thus giving the system the capacity to propose optimal groups of samples (i.e. making the selection optimal as a whole). We propose two methods here: the Iterative method and the Combinatorial method. We prove that both outperform the Sample-focused method.

The second main contribution consists in enforcing an artificial order to the pool of samples. We suggest ordering by distance, to the centroid of either the whole unsupervised pool or the already selected subset as in [28]. We prove that the order helps the SM to find some patterns.

In the end, we get some results that encourage us to follow the proposed ideas since we show that they have an effect on the performance in the considered scenario. However, there is still a need to overcome the stated limitations.

The next logical step should be using an Attention RNN (a concept introduced in [27]), which is a more robust approach that can be better than the enforced artificial order.

6. Acknowledgements

This work has been developed in the framework of project TEC2016-75976-R, financed by the Spanish Ministerio de Economía y Competitividad and the European Regional Development Fund (ERDF)

References

[1] M. Andrychowicz, M. Denil, S. Gómez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. de Freitas. Learning to learn by gradient descent by gradient descent. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3981–3989. Curran Associates, Inc., 2016.

[2] P. Bachman, A. Sordoni, and A. Trischler. Learning algorithms for active learning. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 301–310, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[3] T. Baltrušaitis, C. Ahuja, and L.-P. Morency. Multimodal machine learning: A survey and taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41:423–443, 2017.

[4] K. Cho, B. van Merrienboer, . Glehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. In A. Moschitti, B. Pang, and W. Daelemans, editors, *EMNLP*, pages 1724–1734. ACL, 2014.

[5] T. Collet and O. Pietquin. Active learning for classification: An optimistic approach. 12 2014.

[6] G. Contardo, L. Denoyer, and T. Artières. A Meta-Learning Approach to One-Step Active-Learning. In *International Workshop on Automatic Selection, Configuration and Composition of Machine Learning Algorithms*, volume 1998 of *CEUR Workshop Proceedings*, pages 28–40, Skopje, Macedonia, Sept. 2017. CEUR.

[7] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In D. Precup and Y. W. Teh, editors, *Proc. of the Int. Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[8] C. Finn, K. Xu, and S. Levine. Probabilistic model-agnostic meta-learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 9516–9527. Curran Associates, Inc., 2018.

[9] Y. Freund, H. S. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Mach. Learn.*, 28(2-3):133–168, Sept. 1997.

[10] Y. Gal, R. Islam, and Z. Ghahramani. Deep bayesian active learning with image data. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1183–1192. JMLR.org, 2017.

[11] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines, 2014.

[12] Q. Gu, T. Zhang, J. Han, and C. H. Ding. Selective labeling via error bound minimization. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 323–331. Curran Associates, Inc., 2012.

[13] S. Hoi, R. Jin, J. Zhu, and M. Lyu. Semi-supervised svm batch mode active learning for image retrieval. 06 2008.

[14] G. Koch, R. Zemel, and R. Salakhutdinov. Siamese neural networks for one-shot image recognition. 2015.

[15] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350:1332–1338, 2015.

[16] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. A simple neural attentive meta-learner. In *ICLR*, 2017.

[17] A. Nichol, J. Achiam, and J. Schulman. On first-order meta-learning algorithms. *ArXiv*, abs/1803.02999, 2018.

[18] S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017.

[19] S. Ravi and H. Larochelle. Meta-learning for batch mode active learning. In *ICLR*, 2018.

[20] N. Roy and A. McCallum. Toward optimal active learning through sampling estimation of error reduction. In *ICML*, 2001.

[21] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. P. Lillicrap. One-shot learning with memory-augmented neural networks. *ArXiv*, abs/1605.06065, 2016.

[22] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[23] J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 4077–4087. Curran Associates, Inc., 2017.

[24] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. S. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2017.

[25] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

[26] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.

[27] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc., 2017.

[28] O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. In *International Conference on Learning Representations (ICLR)*, 2016.

[29] O. Vinyals, C. Blundell, T. Lillicrap, k. kavukcuoglu, and D. Wierstra. Matching networks for one shot learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 3630–3638. Curran Associates, Inc., 2016.

[30] M. Wang and W. Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, 2018.

[31] Y. Wang and Q. Yao. Few-shot learning: A survey, 04 2019.

[32] M. Woodward and C. Finn. Active one-shot learning. 02 2017.

[33] J. Yoon, T. Kim, O. Dia, S. Kim, Y. Bengio, and S. Ahn. Bayesian model-agnostic meta-learning. In S. Bengio,

H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 7332–7342. Curran Associates, Inc., 2018.

[34] K. Yu, J. Bi, and V. Tresp. Active learning via transductive experimental design. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 1081–1088, New York, NY, USA, 2006. ACM.