Towards Comparable Active Learning

Thorben Werner [*]
University of Hildesheim
Universitätsplatz 1 31141 Hildesheim
werner@ismll.de

Johannes Burchert[*]
University of Hildesheim
Universitätsplatz 1, 31141 Hildesheim
burchert@ismll.de

Prof. Lars Schmidt-Thieme[*]
University of Hildesheim
Universitätsplatz 1, 31141 Hildesheim
schmidt-thieme@ismll.uni-hildesheim.de

Abstract

In this paper we address the issue of inconsistent results in active learning literature. Authors of previous papers are constantly reporting significant performance improvements, while subsequent literature fails to reproduce those results. This inconsistency leads to a chaotic landscape of AL algorithms. We propose the first AL benchmark that tests algorithms in three major domains of tabular, image and text data. Furthermore, we discuss overlooked problems for reproducing AL experiments with the default seeding setup that depends on a single seed per experiment and provide evidence for the correct amount of repetitions for AL algorithms that reliably converge to the true median performance. We report empirical results for 6 algorithms on 7 datasets and aggregate them into a ranking of AL algorithms via Critical-Difference Diagrams.

1 Introduction

Active Learning (AL) plays an important role in our society that applies machine learning to more and more areas and therefore has a high demand for labeled data in more and more areas. A problem that concerns academic researchers and practitioners in businesses alike and even could be extended to education in schools and hobbyists around the world. On top of providing a principled way to labeled unlabeled datasets, active learning is one of the two major approaches besides semi-supervised learning to make deep learning models more data efficient by requiring only a limited set of manually labeled data. Both approaches are at their core orthogonal and can freely be combined and therefore we should continue our research efforts for both approaches.

Among others, the authors of [18] have pointed out severe inconsistencies in results of AL papers in recent years. In their supplementary materials they conducted a meta analysis of reported results of several different AL algorithms and found that all considered algorithms only provided significant lifts in their own original papers, while all following literature reported performances no better that uncertainty sampling, or in some cases no better than random sampling for the same algorithm. The result of these inconsistencies is a chaotic landscape of AL algorithms where every paper claims to archive state-of-the-art results by significantly outperforming everyone else, while the vast majority of results proves to be non-reproducible.

[*] Institute of Computer Science - Information Systems and Machine Learning Lab (ISMLL)

Submitted to the 37th Conference on Neural Information Processing Systems (NeurIPS 2023) Track on Datasets and Benchmarks. Do not distribute.

1.1 Contributions

1. Evaluation of Active Learning algorithms on datasets from 4 different domains, including synthetic data that highlights principled shortcomings of existing approaches.
2. Novel experimental protocol for seeding the experiment with 3 different seeds to allow full control and reproducibility and analysis of how many restarts are required to converge to the true median performance reliably.
3. Simple algorithm for an Oracle-Curve that can be constructed greedily and does not rely on search.

2 Overview

We constrain our work on pool-based active learning where a pool of unlabeled samples is fixed at the start of each experiment and samples are chosen sequentially. Specifically, we are not experimenting on so-called batch active learning, where at each iteration multiple unlabeled samples are chosen at the same time. Even though batch AL is the more active research domain, it does not have a principled advantage over single-sample AL except speed of computation. Not only is the problem of optimizing a portfolio of unlabeled samples more complicated to solve, the algorithms also have systematically less information per sample to work with. For this reason we propose to focus more research effort on single-sample AL to find better algorithms in an environment that is simpler to solve and easier to control. A performance comparison of batch AL and single-sample AL can be found in Fig. 2, which reproduces the message of Figure 1 from the paper [5] that proposed BALD, one of the SOTA algorithms for AL. We can see that the batched version of BALD [10] can at most perform on-par with the single-sample algorithm. Fig. 2 also serves as a proof of concept for our provided code base. Table 1 shows a feature comparison between our proposed benchmark and several existing benchmarks in the literature, as well as methodological AL papers with experiments on at least two data domains.

Paper	Sampling	# Datasets	Domains	Algorithms	Oracle
Beck et al. [2]	batch	4	1	7	-
Hu et al. [7]	batch	5	2	13	-
Li et al. [11]	batch	5	1	13	-
Zhou et al. [18]	batch	3	2	2	✓
Ours	single	9	4	6	✓

Table 1: Comparison of our benchmark with the existing literature

2.1 Problem Description

Given two spaces $\mathcal{X} := \mathbb{R}^M$ and $\mathcal{Y} := \mathbb{R}^C$ where a samples is drawn from an unknown data distribution $(x, y) \sim p$ with $x \in \mathcal{X}, y \in \mathcal{Y}$. We call $\mathcal{L} := \{(x, y)\}^*$ the labeled pool and $\mathcal{U} := \{x\}^*$ the unlabeled pool. Their domains are $\Omega := \mathbb{R}^M \times \mathbb{R}^C$ and $\Lambda := \mathbb{R}^M$ respectively.
A function $\hat{y} : \mathcal{X} \to \mathcal{Y}$ is called a classifier and a function $\ell : \mathcal{Y} \to \mathcal{Y}$ is called a loss. Given a scalar $B := \mathbb{R};\ B < |\mathcal{U}|$ called budget, we want to find an acquisition function $a : \Omega \times \Lambda \mapsto \mathcal{U}^B$ that creates a subset of $\mathcal{U}$ so that the expected loss on our data is minimal.

$$\mathop{\mathbb{E}}_{(x,y)\sim p} \ell(y, \hat{y}(x))$$

2.2 Lars' Problem Description

Given two spaces $\mathcal{X} := \mathcal{R}^M$ and $\mathcal{Y} := \mathcal{R}^C$, a sample $\mathcal{D}_1, \ldots, \mathcal{D}_N \subseteq (\mathcal{X} \times \mathcal{Y})^*$ of sequences of pairs (x, y) from an unknown distribution p called datasets and a number $B \in \mathcal{N}$ with $B < |\mathcal{D}|$. Given two functions $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathcal{R}$ called loss, and $A : (\mathcal{X} \times \mathcal{Y})^* \times \mathcal{X}^* \to \mathcal{Y}^{\mathcal{X}}$ called learning algorithm, find a function

$$a : (\mathcal{X} \times \mathcal{Y})^* \times \mathcal{X}^* \to \{0,1\}^*$$

called acquisition function, s.t. the expected loss of a model learned on all predictors plus B acquired targets is minimal.

$$\begin{aligned}
&\mathbb{E}_{\mathcal{D}\text{train},\mathcal{D}\text{test}\sim p}\, \text{avg}_{(x,y)\in\mathcal{D}\text{test}}\, \ell(y, \hat{y}(x)) \\
&\quad \text{with } \hat{y} := A((\mathcal{D}\text{train}_{n_1}, \ldots, \mathcal{D}\text{train}_{n_B}), \mathcal{D}\text{train}|_{\mathcal{X}}) \\
&\qquad n_b := \text{index}(a((\mathcal{D}\text{train}_{n_1}, \ldots, \mathcal{D}\text{train}_{n_{b-1}}), \mathcal{D}\text{train}|_{\mathcal{X}})), \quad b \in 1{:}B
\end{aligned}$$

Even though the acquisition function in principle could output the full subset of $\mathcal{D}_{\text{train}}$, the combinatorial problem is computationally not feasible and we allow sequential construction of the subset as a relaxation of the problem.

To construct the active learning setting from existing labeled datasets, we first split it into $\mathcal{D}_{\text{train}}$, $\mathcal{D}_{\text{val}}$ and $\mathcal{D}_{\text{test}}$ and then suppress the labels $\mathcal{Y}$ of $\mathcal{D}_{\text{train}}$ to form the unlabeled pool $\mathcal{U} := \mathcal{X}^*$ and form and initial labeled pool by uniformly sampling k number of instances per class from $\mathcal{U}$ and recovering their label $\mathcal{L} := (\mathcal{X}, \mathcal{Y})^{k*C}$.

Following [18], the quality of an active learning algorithm is evaluated by an ”anytime” protocol that incorporates classification performance at every iteration, not just the final performance after the budget is exhausted. We employ the normalized area under the accuracy curve (AUC):

$$\text{auc}(\mathcal{D}_{test}, \hat{y}, B) := \frac{1}{B}\sum_{i=1}^{B} \text{Acc}(\mathcal{D}_{test}, \hat{y}_i) \tag{1}$$

Where $\hat{y}_i$ is the retrained classification model after the i-*th* instance was selected.

3 Related Work

Version: Braindump

Many different algorithms have been proposed for active learning. In this work we focus on those approaches that have shown consistent results over the years as well as some of the new approaches. AL algorithms can be categorized into two classes: Geometric approaches and uncertainty-based approaches. Geometric approaches use clustering techniques to partition the data and then sample their unlabeled points based on the clusters. They often use the current classification model $\hat{y}_i$ to encode the data into a latent space to improve the performance of their clustering. This benchmark includes the following geometric approaches: CoreSet [15], BADGE [1] and TypiClust [6]. Uncertainty-based approaches use metrics to measure the classifiers state. Commonly, a proxy for the sought after uncertainty of the model for a given datapoint is the distance of that point to the various decision boundaries, measured via the softmax output of the model. This benchmark includes Shannon-Entropy sampling [16], margin sampling [16] and BALD [9]

Some previous work also aimed to provide a benchmark suite for active learning: The authors of [2] and [11] both focus on active learning in the image domain. While [2] discuss a new metric to measure AL performance, which they call ”Label Efficiency” and provide experiments on many common configurations of data preparation, model training and other hyperparameters, [11] focuses on combined approaches of AL and semi-supervised learning to aid model training. The authors of [7] study models that are learned with AL techniques in the image and text domain. They test for several different properties of the models including robustness, response to compression techniques and final performance.

4 Methodology

4.1 Evaluation

We compare different AL algorithms based on their median AUC score (Eq. 1) across multiple restarts of the experiment. Each restart will retain the train/test split (often given by the dataset itself), but introduces a new validation split to mimic the leave-one-out protocol for cross-validation.

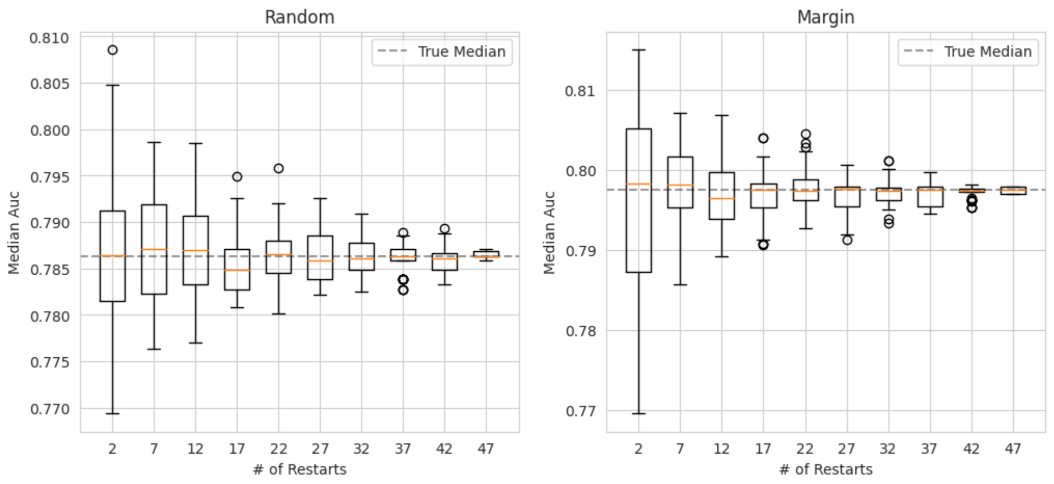


Figure 1: Random draws from an experimental distribution on the Splice datasets with different numbers of repetitions. Each point on the Y-axis represents a cross-validated result that could have been reported in a paper. This analysis shows the drastic differences in performance one could observe even when repeating an experiment 2-10 times.

The AUC incorporates performance in early stages (low budget) as well as capabilities to push the classifier in later stages (high budget). A good AL algorithm should be able to perform well in both scenarios.

Since AUC is dependent on the chosen budget, we need a general rule on how to set this hyperparameter that does not inherently benefit a subset of algorithms. In this work, we choose the budget per dataset to be the first point at which any algorithm (except oracle) manages to reach a percentage of the upper bound performance measured on the full dataset. Even though we would like to propose a single percentage value for all datasets, we found that different data modalities and use cases need different percentages to produce sensible budgets. We propose the following values: **Tabular**: 99%, **Image**: 90% and **Text**: 95%.

Additionally, we provide evidence in Fig. 1 that previous works have not evaluated their experiments with a sufficient number of restarts. To create Fig. 1 we used all our 50 runs from the margin/random sampling algorithm on the splice dataset. From these 50 runs we uniformly sampled subsets of runs and calculated the median AUC for this subset. One AUC value like this corresponds to one cross-validated experiment sampled from the distribution of experiments that are restarted exactly this many times. To create one slice in Fig. 1, we drew 50 samples from this distribution. Each box-plot represents the variance of an evaluation if conducted with the respective number of restarts. We can clearly observe that low repetitions (< 10) provide an uncertain evaluation where lucky and unlucky draws of the same experiment give drastically different median AUC values. To combat is uncertain evaluation, we propose to repeat every experiment 50 times, to arrive at the true median AUC for each algorithm.

4.2 Reproducibility

A big focus in this work is to provide an experimental setup that is fully reproducible independent of the dataset, classification model or AL algorithm used. Given a seed, an evaluation on one dataset should always be done on the same validation split as well as the same random state for all included systems, like the mini batch sampler for model training or the initialization for the classifier itself. Even though different AL algorithms will pick different samples, making them unavailable for sampling in earlier or later batches, the theoretical decision tree for every possible choice in every iteration i should stay the same. Since every possible trajectory cannot be precomputed and stored to disk, we need to resort to seeding. The default choice of setting a global seed at the start of the experiment is not sufficient here, since a single additional random draw from the random number generator completely changes the behavior of all other systems. This additional random number might be drawn during the initialization of the classification model or the AL algorithm, or even during every AL iteration if Ω is stochastic. The desired control only be archived by assigning a separate random number generator to all these processes. To the best of our knowledge, we are the first work that discusses this issue and proposes a solution for it. We hypothesize that the insufficient setup with global seeds contributes to the on-going problem of inconsistent results of AL algorithms in different papers.

In summary, we introduce three different seeds: s_Ω for the acquisition function, $s_\mathcal{D}$ for dataset splitting and mini batch sampling and s_θ for model initialization and sampling of dropout masks. Unless stated otherwise, we will keep s_Ω fixed for restarts of the same experiment, while $s_\mathcal{D}$ and s_θ are incremented by 1 between restarts to introduce stochasticity into our framework.

4.3 Oracle

Posing active learning as a sequence ordering problem, the oracle sequence for a given combination of dataset, model and training procedure would be the sequence that induces the highest AUC score for a given budget. However, since this combinatorial problem is not solvable for realistic datasets, previous works have proposed approximations to this oracle sequence. [18] has used simulated annealing to search for the optimal sequence and used the best solution found after a fixed time budget. Even though their reported performance curves display a significant lift over all other algorithms, we found the computational cost of reproducing this oracle for all our datasets to be prohibitive (The authors reported the search to take several days per dataset on 8 V100 GPUs).

Algorithm 1 Oracle

```
Require: $\mathcal{U}, \mathcal{L}, \mathcal{Y}, \mathcal{D}_{\text{test}}$ Train, Margin, $\tau$, $\hat{y}_\theta$
acc $\leftarrow$ Train$(\mathcal{L}, \mathcal{D}_{\text{test}}, \hat{y}_\theta)$
$r^* \leftarrow 0$
for $t := 1 \dots \tau$ do
    $\mathcal{L}' \leftarrow \mathcal{L}^{(i)} \cup \{(\mathcal{U}_t, y_t)\}$
    acc$' \leftarrow$ Train$(\mathcal{L}', \mathcal{D}_{\text{test}}, \hat{y}_\theta)$
    $r \leftarrow$ acc $-$ acc$'$
    if $r > r^*$ then
        $r^* \leftarrow r$
        $u^* \leftarrow \mathcal{U}_t$
if $r^* = 0$ then
    $u^* \leftarrow$ margin$(\mathcal{U}, \hat{y}_\theta)$
return $u^*$
```

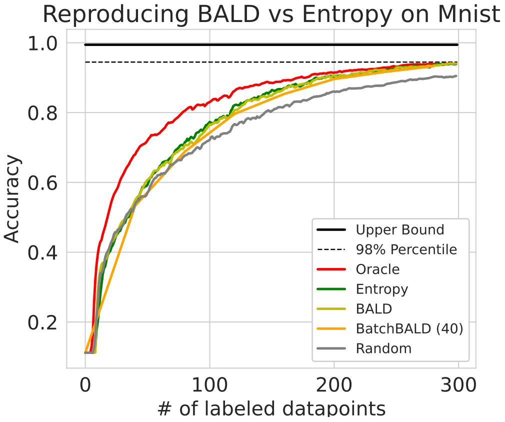


Figure 2: Shows a reproduction of the performance analysis of BALD from the original authors [5] in comparison to entropy sampling and an adaptation of BALD for batch AL [9] with a batch size of 40.

In this paper we propose a greedy oracle algorithm that constructs an approximation of the optimal sequence in an iterative fashion. Our oracle simply tests every data point in the provided sample of unlabeled points by fitting the classifier and directly measuring the resulting validation performance. The point with the best validation performance is selected and added to the labeled pool for that iteration. We noticed that this oracle is overfitting on the validation set, resulting in stagnating or even decreasing performance curves in later AL iterations. To circumvent this problem, we introduced margin sampling as a fallback option for the oracle. Whenever the oracle does not find an unlabeled point that results in an increase in performance (indicating an overfitting position), it defaults to margin sampling in that iteration. The pseudocode for our oracle can be found in Alg. 1. In the

algorithm $\mathrm{Retrain}(\mathcal{L}^{(i)}, \hat{y}_\theta)$ trains the classification model $\hat{y}_\theta$ and returns the accuracy on the test set $\mathcal{D}_{\text{test}}$.
y_t is shorthand for the corresponding label of $u_t^{(i)}$ that can be recovered from the dataset labels.
When the oracle does not find a sample with positive change in classification performance ($r^* = 0$), it applies margin sampling as a fallback ($\mathrm{margin}(u^{(i)}, \hat{y}_\theta)$).
Alg. 1 replaces the acquisition function in the AL process.

5 Implementation Details

5.1 Available Information

At each iteration i the AL algorithm needs to pick an unlabeled datapoint based on a fixed set of information $\{\mathcal{L}^{(i)}, \mathcal{U}^{(i)}, B, |\mathcal{L}^{(i)}| - |\mathcal{L}^{(1)}|, \text{acc}^{(i)}, \text{acc}^{(1)}, \theta^{(i)}, \text{opt}_\theta\}$, where $\theta^{(i)}$ is the current classifier and opt_θ is the optimizer used to fit $\theta^{(i)}$. We allow algorithms to derive additional information of this set like predictions of the classifier, K-Means clustering or even training new classifiers. However, the algorithm may not incorporate external information like other datasets, queries to recover additional labels, or the test/validation set.

5.2 Sampling Strategies

We selected AL algorithms that have good performances reported by multiple different sources. To ensure a fair comparison we fixed the training process of our classification model as well as the set of available information for the algorithm and selected only those that can work under these restrictions:
Uncertainty Sampling Tries to find the sample that the classifier is most uncertain about. For our benchmark we use entropy and margin (a.k.a. best-vs-second-best) sampling.
BALD [9] Applies the query-by-committee strategy of model ensembles to a single model by interpreting the model's parameters as distributions and then sample multiple outputs from them via Monte-Carlo dropout.
BADGE [1] Uses gradient embeddings of unlabeled points to select samples where the classifier is expected to change a lot. The higher the magnitude of the gradient the higher is the expected improvement in model performance.
Coreset [15] Employs K-Means clustering to try to cover all modalities (is this the right word?) that are observed in the data. Selects the unlabeled sample that is the furthest away from all cluster centers. Clustering is done in a semantically meaningful space by encoding the data with the current classifier θ_i. In this work we use the greedy variant of Coreset.
TypiClust [6] Relies on clustering similar to Coreset but proposes a new measure called "Typicality" to select unlabeled centers. Tries to select points that are in the densest regions of clusters that do not contain labeled samples yet. Clustering is done in a semantically meaningful space by encoding the data with the current classifier θ_i. It has to be pointed out that TypiClust was designed for low-budget scenarios, but we think it is still worthwhile to test and compare this algorithm with practically relevant budgets.

5.2.1 Honorable Mentions

Learning Loss for AL Introduces an updated training of the classification model with an auxiliary loss and therefore cannot be compared fairly against classification models without this boosted training regime.

5.3 Choosing the Classifier

Traditionally, the classifier is chosen per dataset so that it is capable of solving the dataset close to the SOTA performance reported in the literature. Since we are not interested in archiving a new SOTA in any classification problem, we opt to use smaller classifiers for the following reasons: Smaller classifiers generally (i) exhibit more stable training behavior and (ii) on average require less sampled datapoints to reach the their upper bound performance on the full dataset. For every dataset the chosen architecture's hyperparameters are optimized by to archive maximum upper bound performance. One desired characteristic of small classifiers is that the ranking of AL algorithms should stay the same when switching to larger models. A small analysis of this behavior can be found in Appendix D. We found that the ranking of AL algorithms unfortunately does change, but we did not observe systematics that benefit one or few specific algorithms. We therefore rely on the different data domains to provide classification models of different sizes and archetypes to cover all of the use-cases. For an overview of architectures and hyperparameters please refer to Appendix C.

5.4 Training the Classifier

The classification model can be trained in two ways. Either you reset the parameters after each AL iteration and train the classifier from scratch with the updated labeled set $\mathcal{L}^{(i)}$, or you retain the previous state and only fine-tune the classifier on $\mathcal{L}^{(i)}$ for a reduced number of epochs. In this work we use the fine-tuning method for raw datasets to save computation, while we use the from-scratch training for embedded dataset, since they have very small classifiers and this method generally produces better results. Our fine-tuning scheme always trains for at least one epoch and employs an aggressive early stopping after that. The early stopping has patience 0, so it will stop as soon as the validation loss does no longer decrease. Even though the use of a fully labeled validation set might be regarded as impractical, since such a set will never exist during deployment, we strongly advocate for using it in benchmarks to control the classifier training. In this work we use the validation set to optimize the hyperparameters of the classifier and reduce overfitting with early stopping the training process in every iteration.

6 Experiments

6.1 Datasets

For all our datasets we use the pre-defined train / test splits, if given. In the remaining cases, we define test sets upfront and store them into separate files to keep them fixed across all experiments. The validation set is split during experiment-time and depends on the dataset-seed.
Tabular: We use **Splice**, **DNA** and **USPS** from LibSVMTools [13]. All three datasets are normalized between [0, 1].
Image: We use **FashionMNIST** [17] and **Cifar10** [10]. Both datasets are normalized according to their standard protocols.
Text: We use **News Category** [12] and **TopV2** [4]. For News Category we use the 15 most common categories as indicated by its Kaggle site. We additionally drop sentences above 80 words to reduce the padding needed (retaining 99,86% of the data). For TopV2, we are only using the ”alarm” domain. Both datasets are encoded with pre-trained GloVe (Common Crawl 840B Tokens) embeddings [14]. Since neither dataset provided a fixed test set, we randomly split 7000 datapoints into a test set.

We would like to point out that these datasets can be considered ”toy-datasets” and therefore not relevant for practical purposes. This might be true if we aimed to develop novel classification models on these datasets, however, similar to our argumentation for picking smaller classifiers, we are solely focused on comparing different AL algorithms in this paper. Our core assumption is that a well-performing algorithm in our benchmark will also transfer into more practical use-cases.

Adapting the experimental setting from [6] we offer all our datasets in the raw setting as well as pre-encoded by a fixed embedding model that was trained by unsupervised contrastive learning. The text datasets are an exception, as they are only offered in their encoded form. The pre-encoded datasets enable us to test our single-sample algorithms on more complex datasets like Cifar10 and FashionMnist without the need of sampling > 2000 datapoints before we can reach our upper bound performance. The embedding model was trained with the SimCLR [3] algorithm. For Cifar10 we adapt the reported hyperparameters from [6] and for the tabular datasets we use random search to optimize the hyperparameters. The quality of embeddings during pretext training was measured af-

	Splice	DNA	USPS
Oracle	0.830 +- 0.01	0.836 +- 0.02	0.823 +- 0.01
SAL	0.799 +- 0.01	0.797 +- 0.03	0.809 +- 0.01
Coreset	0.800 +- 0.01	0.795 +- 0.03	0.787 +- 0.02
TypiClust	0.790 +- 0.01	0.771 +- 0.04	0.761 +- 0.02
MarginScore	0.797 +- 0.02	0.795 +- 0.04	0.808 +- 0.01
ShannonEntropy	0.799 +- 0.02	0.794 +- 0.04	0.807 +- 0.01
RandomAgent	0.788 +- 0.01	0.765 +- 0.03	0.772 +- 0.01
Badge	0.807 +- 0.01	0.769 +- 0.06	0.797 +- 0.02
BALD	0.811 +- 0.01	0.743 +- 0.04	0.717 +- 0.05

Table 2: AUC values for all algorithms on the tabular datasets. Higher is better. Is the STD even helping us here? Since we repeated 50 times, we can use the values as ”true medians”

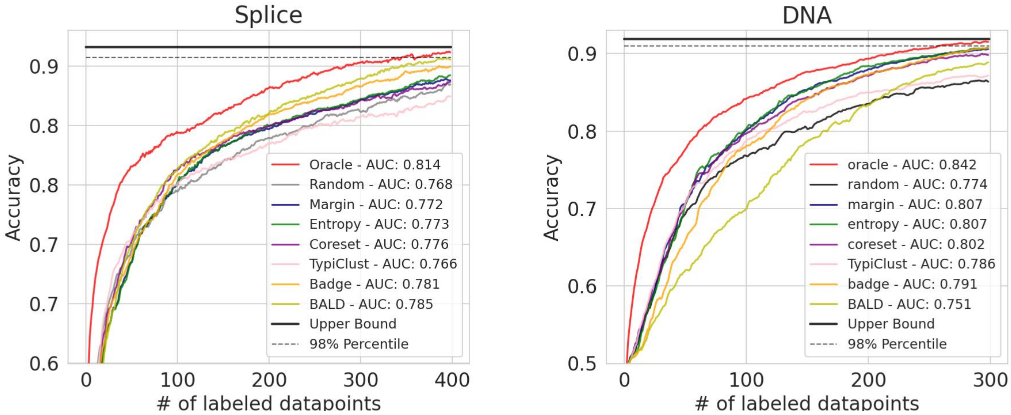


Figure 3: Results for all algorithms on Splice and DNA, both from the tabular domain. Even within one domain, the performance of the same algorithm can vary drastically.

ter each epoch by attaching a linear classification head and evaluating this classifier for test accuracy, mirroring our AL setup for embedded datasets.

6.2 Results

From Fig. 3 we notice drastically different qualities for the same AL algorithm for different datasets. We would like to highlight that both datasets are tabular from the medical domain with similar number of features and classes, yet we see that i.e. BALD is the best algorithm for Splice and the worst algorithm for DNA. These inconsistencies are present between the datasets of all our

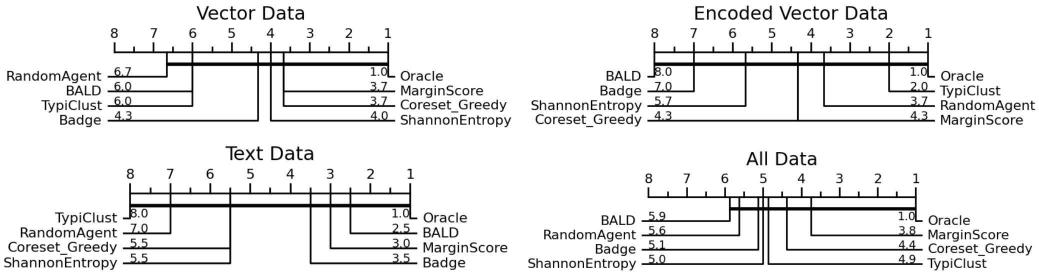


Figure 4: Critical Difference Diagram for all algorithms grouped by domain and all domains combined. Ranks are computed based on median AUC for each algorithm and dataset combination. Lower ranks are better.

tested domains, further highlighting the difficulties for comparing AL algorithms in terms of average performance. In order to provide a meaningful analysis of which algorithm can be expected to perform best on average we ranked the algorithm for each dataset based on their median AUC and displayed these rankings in critical difference diagrams [8]. In Fig. 4 we report the rankings split by domain as well as across all domains (excluding the toydata).

BALD performs bad with linear classifiers since they are trained without dropout and cannot cope well with missing inputs.
TypiClust is better with embedded data not only due to lower budgets. On other datasets it is not able to outperform other algorithms in early stages

7 Conclusion

8 Limitations and Future Work

No batch AL
No learned algorithms
No SOTA classifier training (data augmentation, semi-supervised, etc.)
SimCRL as Pretext task works better for images

Acknowledgments and Disclosure of Funding

Funded by the Lower Saxony Ministry of Science and Culture under grant number ZN3492 within the Lower Saxony “Vorab“ of the Volkswagen Foundation and supported by the Center for Digital Innovations (ZDIN).

References

[1] Jordan T Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal. Deep batch active learning by diverse, uncertain gradient lower bounds. In *International Conference on Learning Representations*, 2020.

[2] Nathan Beck, Durga Sivasubramanian, Apurva Dani, Ganesh Ramakrishnan, and Rishabh Iyer. Effective evaluation of deep active learning on image classification tasks. *arXiv preprint arXiv:2106.15324*, 2021.

[3] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[4] Xilun Chen, Asish Ghoshal, Yashar Mehdad, Luke Zettlemoyer, and Sonal Gupta. Low-resource domain adaptation for compositional task-oriented semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Association for Computational Linguistics, 2020.

[5] Yarin Gal, Riashat Islam, and Zoubin Ghahramani. Deep bayesian active learning with image data. In *International conference on machine learning*, pages 1183–1192. PMLR, 2017.

[6] Guy Hacohen, Avihu Dekel, and Daphna Weinshall. Active learning on a budget: Opposite strategies suit high and low budgets. *arXiv preprint arXiv:2202.02794*, 2022.

[7] Qiang Hu, Yuejun Guo, Maxime Cordy, Xiaofei Xie, Wei Ma, Mike Papadakis, and Yves Le Traon. Towards exploring the limitations of active learning: An empirical study. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 917–929. IEEE, 2021.

[8] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[9] Andreas Kirsch, Joost Van Amersfoort, and Yarin Gal. Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning. *Advances in neural information processing systems*, 32, 2019.

[10] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[11] Yu Li, Muxi Chen, Yannan Liu, Daojing He, and Qiang Xu. An empirical study on the efficacy of deep active learning for image classification. *arXiv preprint arXiv:2212.03088*, 2022.

[12] Rishabh Misra. News category dataset. *arXiv preprint arXiv:2209.11429*.

[13] Information Engineering Graduate Institute of Taiwan University. Libsvmtools.

[14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[15] Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. *arXiv preprint arXiv:1708.00489*, 2017.

[16] Dan Wang and Yi Shang. A new active labeling method for deep learning. In *2014 International joint conference on neural networks (IJCNN)*, pages 112–119. IEEE, 2014.

[17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[18] Yilun Zhou, Adithya Renduchintala, Xian Li, Sida Wang, Yashar Mehdad, and Asish Ghoshal. Towards understanding the behaviors of optimal deep active learning algorithms. In *International Conference on Artificial Intelligence and Statistics*, pages 1486–1494. PMLR, 2021.

A Other result tables

	SpliceEncoded	DNAEncoded	USPSEncoded
Oracle	0.754 +- 0.02	0.726 +- 0.02	0.674 +- 0.01
SAL	0.675 +- 0.03	0.640 +- 0.04	0.634 +- 0.01
Coreset	0.690 +- 0.02	0.644 +- 0.05	0.607 +- 0.02
TypiClust	0.695 +- 0.02	0.660 +- 0.03	0.643 +- 0.01
MarginScore	0.675 +- 0.03	0.643 +- 0.05	0.632 +- 0.02
ShannonEntropy	0.673 +- 0.03	0.638 +- 0.05	0.626 +- 0.02
RandomAgent	0.680 +- 0.03	0.633 +- 0.04	0.594 +- 0.02
Badge	0.670 +- 0.04	0.600 +- 0.07	0.597 +- 0.02
BALD	0.660 +- 0.04	0.597 +- 0.06	0.652 +- 0.01

Table 3: AUC values for all algorithms on the encoded tabular datasets. Higher is better.

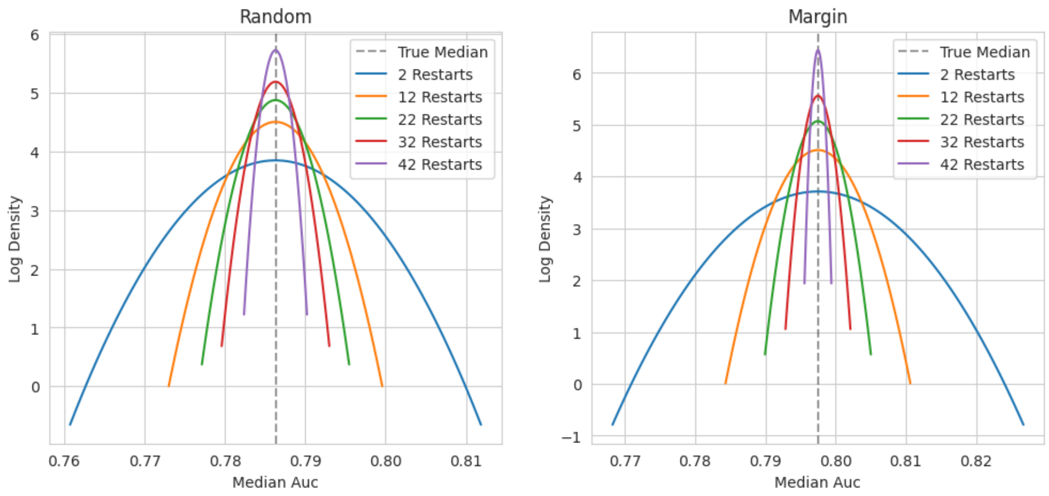
	Cifar10Encoded	FashionMnistEncoded
Oracle	0.699 +- 0.01	0.721 +- 0.01
SAL	0.623 +- 0.01	0.667 +- 0.01
Coreset	0.641 +- 0.01	0.674 +- 0.01
TypiClust	0.627 +- 0.01	0.638 +- 0.01
MarginScore	0.641 +- 0.01	0.676 +- 0.01
ShannonEntropy	0.637 +- 0.02	0.680 +- 0.01
RandomAgent	0.598 +- 0.02	0.633 +- 0.01
Badge	0.645 +- 0.02	0.681 +- 0.01
BALD	0.633 +- 0.01	0.666 +- 0.01

Table 4: AUC values for all algorithms on the encoded image datasets. Higher is better.

	TopV2	News
Oracle	0.860 +- 0.01	0.448 +- 0.01
SAL	0.831 +- 0.01	0.312 +- 0.01
Coreset	0.816 +- 0.02	0.355 +- 0.02
TypiClust	0.718 +- 0.02	0.314 +- 0.01
MarginScore	0.821 +- 0.02	0.357 +- 0.01
ShannonEntropy	0.794 +- 0.02	0.342 +- 0.01
RandomAgent	0.776 +- 0.02	0.349 +- 0.01
Badge	0.824 +- 0.01	0.343 +- 0.01
BALD	0.825 +- 0.02	0.347 +- 0.01

Table 5: AUC values for all algorithms on the text datasets. Higher is better.

B Alternative Plot for Restarts Ablation



C Hyperparameters per Dataset

Dataset	Classifier	Optimizer	LR	Weight Decay	Dropout
Splice	[24, 12]	NAdam	1.2e-3	5.9e-5	0
SpliceEnc.	linear	NAdam	6.2e-4	5.9e-6	0
DNA	[24, 12]	NAdam	3.9e-2	3.6e-5	0
DNAEnc	linear	NAdam	1.6e-3	4e-4	0
USPS	[24, 12]	Adam	8.1e-3	1.5e-6	0
USPS	linear	NAdam	7.8e-3	1.9e-6	0
FashionMnist	ResNet18	NAdam	1e-3	0	0
FashionMnistEnc	linear	Adam	1.6e-3	1e-5	5e-2
Cifar10	ResNet18	NAdam	1e-3	0	0
Cifar10Enc	linear	NAdam	1.7e-3	2.3e-5	0
TopV2	BiLSTM	NAdam	1.5e-3	1.7e-7	5e-2
News	BiLSTM	NAdam	1.5e-3	1.7e-7	5e-2

Table 6: Classifier architectures and optimized hyperparameters per dataset. Numbers in brackets signify a MLP with corresponding hidden layers.

D Comparison of Different Classifier Sizes

We tested two different classifier sizes in Splice and DNA:

- Small: [24, 12] (2400 parameters)
- Big: [24, 48, 48] (5700 parameters)

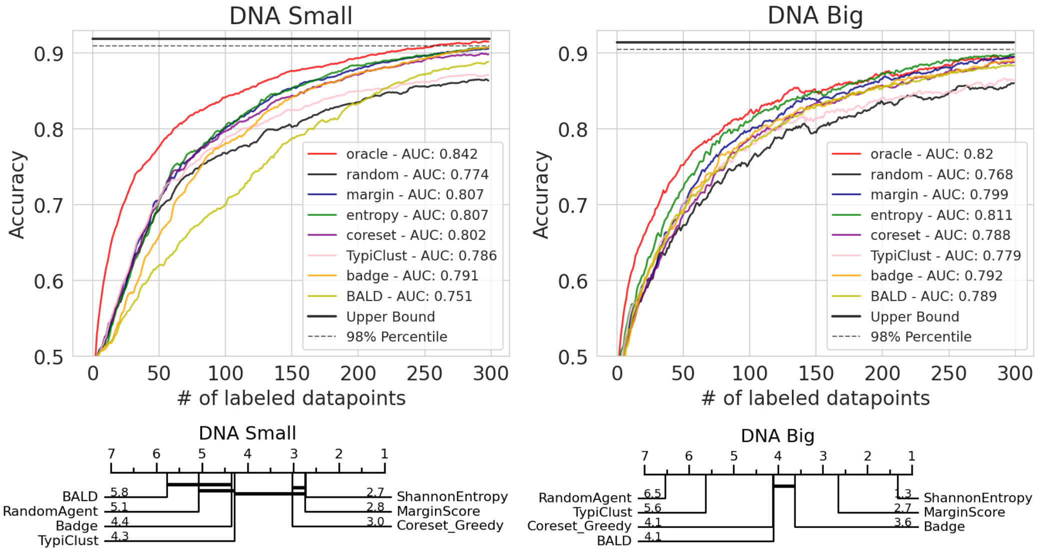


Figure 5: Comparison of small and big classifiers for the DNA dataset

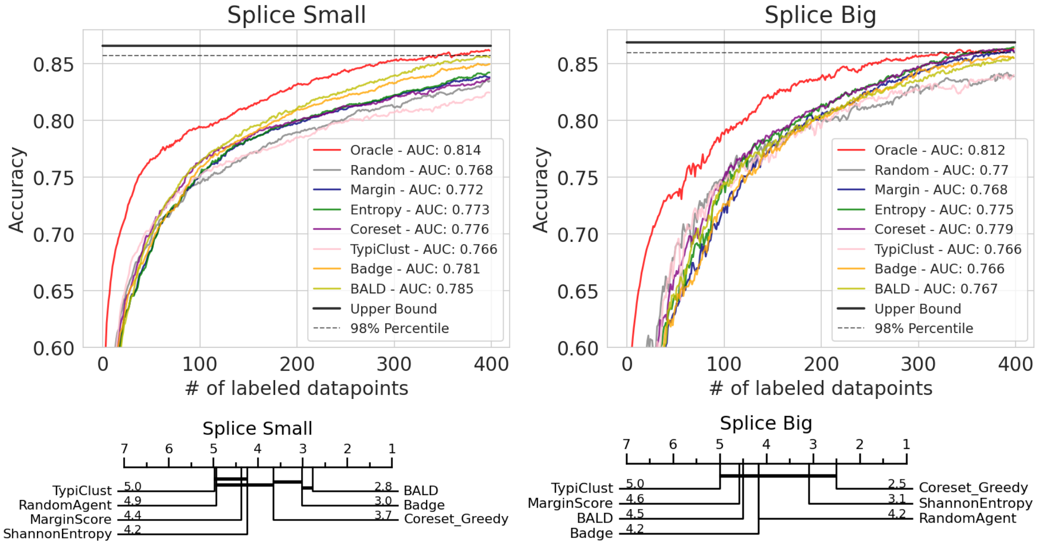
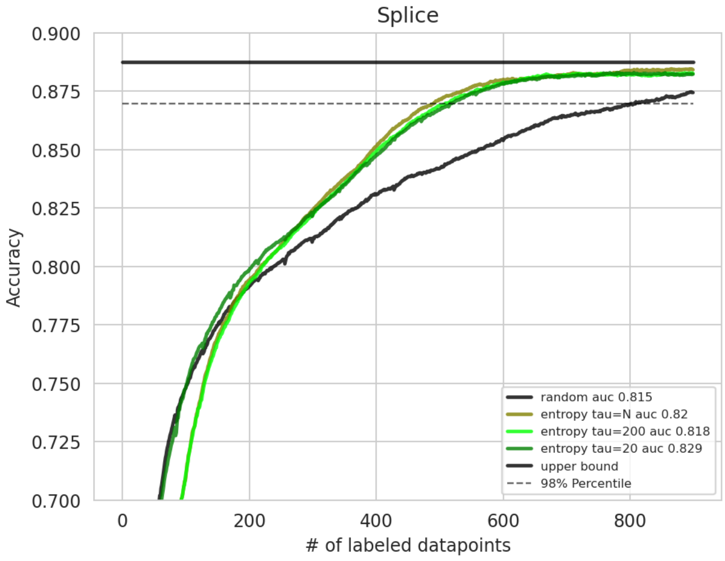


Figure 6: Comparison of small and big classifiers for the Splice dataset

E Comparison of different sample sizes



F AL Pseudocode

Algorithm 2 Active Learning

```
Require: $\mathcal{U}$                                  ▷ Unlabeled Pool
Require: $\tau$                                         ▷ Unlabeled Sample Size
Require: $\Omega$                                       ▷ AL Agent
Require: $\omega$                                       ▷ Environment State function
$\mathcal{L}^{(1)} \leftarrow \text{seed}(\mathcal{U})$                ▷ Create the initial labeled set
$\mathcal{U}^{(1)} \leftarrow \mathcal{U}$
for $i := 1 \dots B$ do
    $\text{acc}^{(i)} \leftarrow \text{Retrain}(\mathcal{L}^{(i)})$
    $a^{(i)} \leftarrow \Omega(\mathcal{U}^{(i)})\ ; \quad a \in 1 : |\mathcal{U}^{(i)}|$
    $y^{(i)} \leftarrow \text{label}(\mathcal{U}_a^{(i)})$
    $\mathcal{L}^{(i+1)} \leftarrow \mathcal{L}^{(i)} \cup \{(\mathcal{U}_a^{(i)}, y^{(i)})\}$
    $\mathcal{U}^{(i+1)} \leftarrow \mathcal{U}^{(i)} \setminus \{\mathcal{U}_a^{(i)}\}$
return $\frac{1}{B} \sum_{i=1}^{B} \text{acc}^{(i)}$
```

Algorithm 3 Retrain

```
Require: $\mathcal{L}$                                        ▷ Labeled Pool
Require: $\mathcal{D}_{\text{val}}$                           ▷ Validation Data
Require: $\mathcal{D}_{\text{test}}$                          ▷ Test Data
Require: $\hat{y}_\theta$                                     ▷ Class. Model
Require: $e^{\text{max}}$                                     ▷ Maximum Epochs
loss$^* \leftarrow \infty$
for $i := 1 \ldots e^{\text{max}}$ do
    $\theta_{i+1} \leftarrow \theta_i - \eta \nabla_\theta \ell(\mathcal{L}, \hat{y}_\theta)$
    loss$_i \leftarrow \ell(\mathcal{D}^{\text{val}}, \hat{y}_\theta)$
    if loss$_i <$ loss$^*$ then
        loss$^* \leftarrow$ loss$_i$
    else
        Break
return Acc($\mathcal{D}^{\text{test}}, \hat{y}_\theta$)
```