Towards Comparable Active Learning

Anonymous Authors

Abstract

Active Learning has received significant attention in the field of machine learning for its potential in selecting the most informative samples for labeling, thereby reducing data annotation costs. However, the lack of reproducibility of performance gains reported in recent literature has created a chaotic landscape in Active Learning research. This paper addresses these issues of inconsistent results in active learning (AL) literature. To the best of our knowledge, we propose the first AL benchmark that tests algorithms in 3 major domains: Tabular, Image and Text. Furthermore, we highlight overlooked problems for reproducing AL experiments that can lead to unfair comparisons and increased variance in the results. To tackle these challenges, we propose an experimental protocol to accurately control the experiments. We report empirical results for 6 widely used algorithms on 7 datasets and aggregate them into a domain-specific ranking of AL algorithms.

1 Introduction

Deep neural networks (NN) have produced state-of-the-art results on many important supervised learning tasks. Since Deep NNs usually require large amounts of labeled training data, Active Learning (AL) can be used to instead to select the most informative samples out of a large pool of unlabeled data, so that only these samples need to be labeled. It has been shown that a small labeled set of this nature can be used to train well-performing models [2, 7, 11, 21].

In the last decade many different algorithms for AL have been proposed. Even though, almost every method has reported lifts all it's predecessors [1] , AL research faces two central difficulties: (i) The experiments are often carried out on different datasets and model architectures, hindering direct comparison, (ii) the reported results prove to be very difficult to reproduce. While multiple benchmark suites have been proposed to solve (i), to the best of our knowledge, we are the first to compare AL algorithms on all 3 data domains of vector, image and text. Regarding (ii), [21] has pointed out severe inconsistencies in results of AL papers in recent years. 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 following literature reported performances no better that uncertainty sampling, or in some cases no better than random sampling for the same algorithm ([21] Appendix A). The result of these inconsistencies is a chaotic landscape of AL algorithms where every paper claims to archive state-of-the-art performance , while the vast majority of results proves to be non-reproducible.

In this work we propose an evaluation protocol that was designed to cope with the high variance in the performances of AL algorithms as well as being fully controllable regardless of the combination of dataset, model and AL algorithm.

[1] out of all considered algorithms for this paper, only BALD [5] did not claim a new SOTA performance in their result section

Code available at: `https://github.com/wernerth94/comparable-active-learning`

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

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. [21]	batch	3	2	2	✓
Ours	single	7	3	6	✓

Table 1: Comparison of our benchmark with the existing literature. Oracle curves serve as an approximation of the best possible AL algorithm.

We focus our work on single-sample pool-based AL 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 most proposed algorithms (and benchmarks) are batched, they do not have a principled advantage over single-sample AL except speed of computation. The problem of optimizing a portfolio of unlabeled samples in each iteration is more complicated to solve and the algorithms have systematically less information per sample to work with. A performance comparison of batch AL and single-sample AL can be found in Fig. 1. We can see that for two established algorithms BALD [10] and BADGE [1] Batch AL can at most perform on-par with single-sample AL, independent of wether algorithm was originally designed for batch AL or not.
Table 1 shows a feature comparison between our proposed benchmark and several existing benchmarks in the literature.

Contributions

1. Evaluation of Active Learning algorithms on datasets from 3 different domains
2. Two novel synthetic dataset that highlight principled shortcomings of existing AL algorithms
3. Novel experimental protocol for controlling AL experiments to allow full reproducibility
4. Simple algorithm for an Oracle that can be constructed greedily and does not rely on search

2 Problem Description

Given two spaces $\mathcal{X}, \mathcal{Y}$, $n = l + u$ data points with $l \in \mathbb{N}$ labeled examples $\mathcal{L} = \{(x_1, y_1), \ldots, (x_l, y_l)\}$, $u \in \mathbb{N}$ unlabeled examples $\mathcal{U} = \{x_{l+1}, \ldots, x_n\}$, a budget $\mathbb{N} \ni b \leq u$ and an annotator $A : \mathbb{R}^{\mathcal{X}} \to \mathbb{R}^{\mathcal{Y}}$ that can label x. We call $x \in \mathcal{X}$, $y \in \mathcal{Y}$ predictors and labels respectively where (x, y) are drawn from an unknown distribution ρ. Find an acquisition function $\Omega : \mathcal{U}^{(i)}, \mathcal{L}^{(i)} \mapsto x^{(i)}$ that iteratively selects the next unlabeled point $x^{(i)} \in \mathcal{U}^{(i)}$ for labeling

$$\mathcal{L}^{(i+1)} \leftarrow \mathcal{L}^{(i)} \cup \{\Big(x^{(i)}, A(x^{(i)})\Big)\}$$
$$\mathcal{U}^{(i+1)} \leftarrow \mathcal{U}^{(i)} \setminus x^{(i)}$$

so that the expected loss $\ell : \mathbb{R}^{\mathcal{Y}} \times \mathbb{R}^{\mathcal{Y}} \to \mathbb{R}$ of the machine learning algorithm $\hat{y}$ after B iterations is minimal:

$$\min \quad \mathbb{E}_{(x,y)\sim\rho} \ell(y, \hat{y}(\mathcal{L}^{(B)}))$$

This formulation is a special case of the problem formulation in Appendix A.

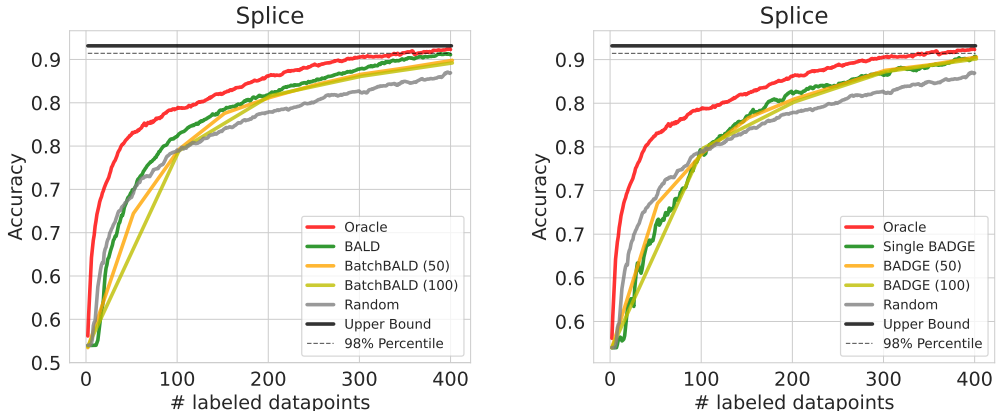


Figure 1: Performance comparison of BALD [5] with it's batch AL adaptation [8] and BADGE [1] with it's single-sample AL adaptation ($k = 1$ in [1] Alg. 1). In either scenario, the batched algorithm can at most perform on-par with it's single-sample analog, independent of wether it was originally designed for batch AL.

3 Related Work

Multiple benchmark suites have already been proposed for Active Learning: The authors of [2] and [11] both focus exclusively on Batch AL 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. The authors of [7] study models that are trained 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. [21] proposed an oracle algorithm for AL that uses Simulated Annealing search to approximate a solution for the optimal subset of labeled data. Additionally, they study the generalization behavior of subsets of labeled data in the text an image domain. The employed AL algorithms for our experiments are introduced in Section 4.4.

4 Methodology

4.1 Evaluation

Following [21], the quality of an AL algorithm is evaluated by an "anytime" protocol that incorporates classification performance at every iteration, as opposed to evaluating final performance after the budget is exhausted. We employ the normalized area under the accuracy curve (AUC):

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

where $\hat{y}_i$ is the (re-)trained classification model after the i-*th* iteration. To mimic the leave-one-out protocol for cross-validation we will restart each experiment multiple times. Each restart will retain the train/test split (often given by the dataset itself), but introduces a new validation split. The AUC incorporates performance in early stages (low budget) as well as capabilities to push the classifier in later stages (high budget). AL algorithms have to perform well in both scenarios. Since AL performance inhibits high variance and is prone to outliers, we propose to aggregate AUC values with their median instead of mean.

Since the AUC depends 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

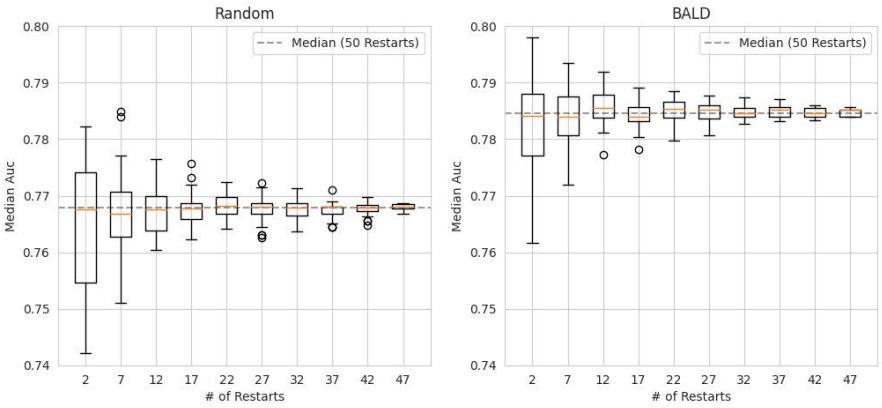


Figure 2: Random draws from an experimental distribution on the Splice dataset with different numbers of repetitions. Each point on the Y-axis represents a cross-validated result that could have been reported in a paper. This analysis shows the drastic differences in performance one could observe even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

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%, **Text**: 95%, **Embedded Data**: 90%.

Additionally, we provide evidence in Fig. 2 that previous works have not evaluated their experiments with a sufficient number of restarts. To create Fig. 2 we used 50 restarts from the margin/random acquisition function on the Splice dataset. From these 50 runs we uniformly sampled subsets of runs and calculated the median AUC for this subset. One of these median AUC values 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. 2, 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 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 reliably arrive at the true median AUC, we propose to repeat every experiment 50 times, as only > 42 repetitions don't produce outliers anymore (as indicated by the rightmost columns in Fig 2). One way to reduce the number of necessary repetitions would be to reduce the amount of variance in the experiment through specialized seeding (discussed in the next section). We ultimately decided in favor of high variance and high number of repetitions as the high variance accurately reflects real world applications of AL.

4.2 Reproducibility

Previous works have noted adverse effects of training stochasticity on the evaluation of AL algorithms ([21, 12]). Both papers observed that an actively sampled labeled set does not generalize well between model architectures or even different initializations of the same model. To cater for this we aim to provide an experimental setup that is fully reproducible independent of the dataset, classification model or AL algorithm used. For a fair comparison of two AL algorithms, both algorithms need to receive equal starting conditions in terms of train / validation split, initialization of classifier and even the state of minor systems like the optimizer or mini-batch sampler. Even though different implementations might have their own solution to some of these problems, to the best of our knowledge no previous work has discussed this topic in detail. The main obstacle for

Since we only consider single-sample AL, increasing the budget comes at a high computational cost. Therefore we could not freely increase the budget above 2000.

ensuring reproducibility is the seeding utility in PyTorch, Tensorflow and other frameworks, whose default choice is a single global seed. Since many systems draw random numbers from this seed, all of them influence each other to a point where a single additional draw can completely change the model initialization or data split, depending on the ordering of these two in the implementation. Even though some workarounds exists, like re-setting the seed multiple times, this problem is not limited to the initialization phase, but also extends to the AL iterations and the systems within. We propose an implementation that creates a separate Random Number Generator (RNG) for each of these systems to ensure equal testing conditions even when the AL algorithm, dataset or classifier changes. 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 AL algorithm, $s_\mathcal{D}$ 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. Some algorithms require a subsample to be drawn from $\mathcal{U}$ in order to reduce the computational cost in each iteration, while others need access to the full unlabeled pool (i.e. for effective clustering). If a subsample is required, it will be drawn from s_Ω and therefore will not influence other systems in the experiments. For each algorithm, we decided if subsampling is required based on our available hardware, but decided against setting a fixed time limit per experiment, since this would introduce unnecessary complexity into the benchmark. An overview of selected hyperparameters per AL algorithm can be found in Appendix D.

4.3 Oracle

Posing Active Learning as a combinatorial problem, the oracle set $\mathcal{U}_b$ for a given dataset, model and training procedure is the set that induces the highest AUC score for a given budget. However, since this problem is not solvable for realistic datasets, previous works have proposed approximations to this oracle sequence. [21] has used simulated annealing to search for the optimal subset and used the best solution found after a fixed time budget. Even though their reported performance curves display a significant lift over all other acquisition functions, 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). In this paper we propose a greedy oracle algorithm that constructs an approximation of the optimal set in an iterative fashion.

Algorithm 1 Oracle

Require: $\mathcal{U}, \mathcal{L}, \mathcal{Y}, \mathcal{D}_{\text{test}}$ Train, Margin, $\tau, \hat{y}_\theta$

acc $\leftarrow \text{Train}(\mathcal{L}, \mathcal{D}_{\text{test}}, \hat{y}_\theta)$
$r^* \leftarrow 0$
for $k := 1 \dots \tau$ **do**
  $\mathcal{L}' \leftarrow \mathcal{L}^{(i)} \cup \{(u_k, A(u_k))\}$
  acc$' \leftarrow \text{Train}(\mathcal{L}', \mathcal{D}_{\text{test}}, \hat{y}_\theta)$
  $r \leftarrow$ acc $-$ acc$'$
  if $r > r^*$ **then**
    $r^* \leftarrow r$
    $u^* \leftarrow u_k$
if $r^* = 0$ **then**
  $u^* \leftarrow \text{margin}(\mathcal{U}, \hat{y}_\theta)$
return u^*

Our oracle simply evaluates every data point $u_k = \text{unif}(\mathcal{U}|\text{k} = 1 \dots \tau)$ in a subsample of unlabeled points by fitting the classifier $\hat{y}$ on $\mathcal{L}^{(i)} \cup u_k$ and directly measuring the resulting test performance. The data point with the best test performance is selected and added to the labeled pool for that iteration. We noticed that this oracle is overfitting on the test set, resulting in stagnating or even decreasing performance curves in later AL iterations. This can happen for example, if the oracle picked a labeled set that enables the classifier to correctly classifier a big portion of easy samples in the test set, but now fails to find the next single unlabeled point that would enable the classifier to succeed on one of the hard samples in the test set. This leads to a situation, where the selected data point is basically random.

To circumvent this problem, we introduced margin sampling [19] as a fallback option for the oracle. Whenever the oracle does not find an unlabeled point that results in an increase in performance, it defaults to margin sampling in that iteration. The resulting greedy algorithm constructs an approximation of the optimal labeled set that consistently outperforms all other algorithms by a significant margin, while requiring relatively low computational cost (scaling linearly in τ). The pseudocode

for our oracle can be found in Alg. 1. In the algorithm $\mathrm{Train}(\mathcal{L}, \mathcal{D}_{\text{test}}, \hat{y}_\theta)$ trains the classification model $\hat{y}_\theta$ and returns it's accuracy on $\mathcal{D}_{\text{test}}$.
Alg. 1 replaces the acquisition function in the AL process.

4.4 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 algorithms and selected only those that can work under these restrictions:
Uncertainty Sampling Tries to find the sample that the classifier is most uncertain about by computing heuristics of the class probabilities. For our benchmark we use entropy and margin (a.k.a. best-vs-second-best) sampling.
BALD [8] Applies the query-by-committee strategy of model ensembles to a single model by interpreting the classifier'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 [16] Employs K-Means clustering to try to cover the whole data distribution. 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 $\hat{y}$. 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 samples. Selects 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 $\hat{y}$. 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.

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.

4.5 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 these 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 F. We found that the ranking of AL algorithms unfortunately does change slightly, 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 E.

5 Experiments

5.1 Implementation Details

At each iteration i the AL algorithm picks an unlabeled datapoint based on a fixed set of information $\{\mathcal{L}^{(i)}, \mathcal{U}^{(i)}, B, |\mathcal{L}^{(i)}| - |\mathcal{L}^{(1)}|, \text{acc}^{(i)}, \text{acc}^{(1)}, \hat{y}^{(i)}, \text{opt}_{\hat{y}}\}$, where $\hat{y}^{(i)}$ is the current classifier and $\text{opt}_{\hat{y}}$ is the optimizer used to fit $\hat{y}^{(i)}$. We allow acquisition functions 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, additional training steps for $\hat{y}$, or the test/validation set.

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.

5.2 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 [14]. All three datasets are normalized between [0, 1].

Image: We use **FashionMNIST** [20] and **Cifar10** [10]. Both datasets are normalized according to their standard protocols.

Text: We use **News Category** [13] 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 [15]. 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 Fashion-Mnist. 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 after each epoch by attaching a linear classification head and evaluating this classifier for test accuracy, mirroring our AL setup for embedded datasets.

Every dataset has a fixed size for the seed set of 1 sample per class, with the only exceptions being

	Vector	Vector Enc.	Image	Image Enc.	Text	All
Oracle	1.07 ± 0.02	1.13 ± 0.01	1.07 ± 0.04	1.16 ± 0.02	1.16 ± 0.07	1.12 ± 0.05
Margin	1.03 ± 0.02	1.03 ± 0.03	1.01 ± 0.01	1.07 ± 0.00	1.04 ± 0.00	1.04 ± 0.02
Badge	1.02 ± 0.01	0.98 ± 0.04	1.01 ± 0.00	1.07 ± 0.00	1.04 ± 0.01	1.02 ± 0.04
Entropy	1.03 ± 0.02	0.99 ± 0.01	1.00 ± 0.01	1.07 ± 0.00	1.02 ± 0.00	1.02 ± 0.03
Coreset	1.02 ± 0.01	0.99 ± 0.04	1.00 ± 0.00	1.07 ± 0.00	1.03 ± 0.02	1.02 ± 0.04
TypiClust	1.03 ± 0.03	1.05 ± 0.04	0.99 ± 0.02	1.03 ± 0.01	0.98 ± 0.04	1.02 ± 0.04
Random	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
BALD	0.98 ± 0.03	0.90 ± 0.04	0.99 ± 0.03	1.06 ± 0.01	1.04 ± 0.01	0.98 ± 0.07

Table 2: Performances for each algorithm per domain normalized by the performance of random sampling. Higher is better, scores below 1 indicate worse-than-random performance.

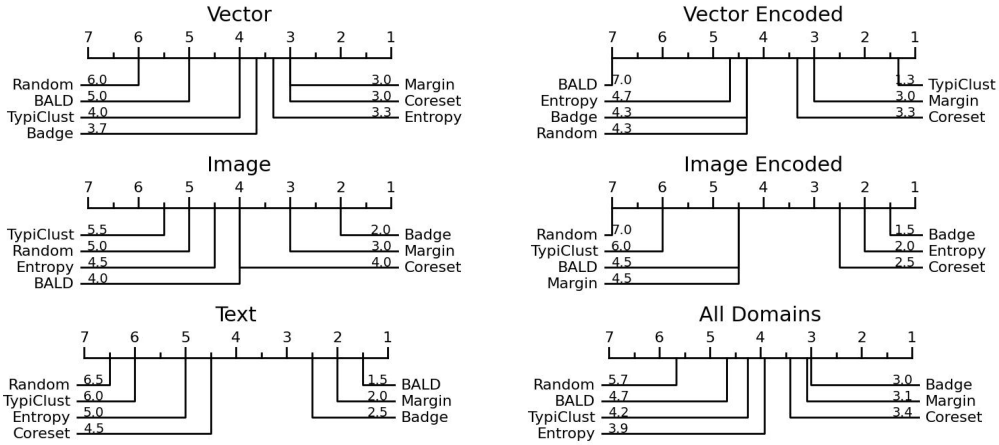


Figure 3: Critical Difference Diagram for all algorithms grouped by domain and all domains combined. Ranks are computed based on median AUC for each algorithm and dataset combination. Lower ranks are better.

raw FashionMnist and Cifar10 with 100 examples per class to alleviate the cold-start problem in these complex domains.

5.3 Results

We report our results in two different fashions: Table 2 contains mean AUC performances per data domain. To make performances from different datasets comparable we normalized them with the random performance of each dataset. Figure 3 aggregates the performances per domain by counting wins and deriving a ranking of algorithms. This is closely related to the evaluation protocol of [1], who displayed a pairwise penalty matrix for all algorithms.

Based these two evaluations we make the following observations:

(i) The ranking of both reports does not match exactly. This is due to Table 2 being more sensitive to outlier performances (i.e. BALD being worse than random due to it's bad performance on embedded vector data).

(ii) Both evaluations make it clear that no one-fits-all algorithm for all domains seems to exist, with only margin sampling being significantly better that random in Tab. 2 and the best 3 ranks for all domains in Fig. 3 being close together.

(iii) Even though in Fig. 3 some domains show evidence of one algorithm being systematically better than the rest (i.e. Image), the actual differences in Tab. 2 seem to be negligible.

(iv) Not only do different domains favor different AL algorithms, we noticed that even within the same domain datasets can induce drastically different behaviors (Compare Appendix B Splice/DNA/USPS).

6 Discussion

Our results empirically reinforce what the authors of [21] have noticed in their meta-analysis already: When slightly altering the dataset, model or training conditions, most AL algorithms are reduced to uncertainty sampling or worse. Additionally, some results remain non-reproducible even on the same dataset and model combination (Compare Appendix B Cifar10 with [6] or [16]). On the other hand we were able to also confirm some previous results i.e. for the NewsCategory dataset (Compare Appendix B News with [7] Fig 3g).

Regarding meta analysis of our results we have the following conclusions:
(i) BALD seems to be performing especially bad when the input dimensionality is low and the applied classifier is too simple. (i.e. (Encoded) Vector).
(ii) TypiClust seemingly holds its claim to work well with extremely low budgets (Encoded Vector $B = \{50, 60, 600\}$), but fails to push the classifier close to upper bound performance even with moderate budgets in other domains.
(iii) The authors of [1] noted that diversity sampling and adjacent methods, like Coreset and TypiClust, only work well when the classifier has correct inductive biases (correct architecture, tuned hyperparameters). Since our architectures are extensively tuned, the performance of these algorithms might be slightly inflated.

7 Limitations and Future Work

In the current state, a couple of aspects/use-cases are missing from our proposed benchmark and are considered future work:
(i) We elected to not provide any experiments on batch AL, but the provided implementation is already capable of doing that and we plan to extend the benchmark to include batch sizes larger than 1 in our experiments.
(ii) Due to difficulties with reproducing results and high computational setup costs, we did not include learned acquisition functions like [17], [18] or [9]
(iii) Our training process for the classifier does not include techniques that would be present in real-world scenarios, like data augmentation, semi-supervised learning, or auxiliary tasks. We opted for simpler training process to be able to extensively tune those fewer hyperparameters and have full control over the training.
(iv) We used SimCLR as pretext task for our vector datasets, which primarily relies on data augmentation. While images have a plethora of well-defined augmentation techniques, we had to rely on Gaussian noise to augment the vector data. Other pretext tasks, like autoencoders have potential to provide better encodings of the data.

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] 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.

[9] Ksenia Konyushkova, Raphael Sznitman, and Pascal Fua. Learning active learning from data. *Advances in neural information processing systems*, 30, 2017.

[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] David Lowell, Zachary C Lipton, and Byron C Wallace. Practical obstacles to deploying active learning. *arXiv preprint arXiv:1807.04801*, 2018.

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

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

[15] 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.

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

[17] Jingyu Shao, Qing Wang, and Fangbing Liu. Learning to sample: an active learning framework. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 538–547. IEEE, 2019.

[18] Thuy Vu, Ming Liu, Dinh Phung, and Gholamreza Haffari. Learning how to active learn by dreaming. In *Proceedings of the 57th annual meeting of the Association for Computational Linguistics*, pages 4091–4101, 2019.

[19] 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.

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

[21] 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 Problem Formulation

Given

- a number $B \in \mathbb{N}$ (called budget),
- two spaces $\mathcal{X}$ and $\mathcal{Y}$, e.g., $\mathcal{X} := \mathbb{R}^M$, $\mathcal{Y} := \mathbb{R}^T$,
- a sample $\mathcal{D}_1, \ldots, \mathcal{D}_N \in (\mathcal{X} \times \mathcal{Y})^*$ of sequences of pairs (x, y) from an unknown distribution p (called datasets), with $p(\mathcal{D}) = 0$ for $|\mathcal{D}| < B$,
- a function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ (called loss), and
- a function $\hat{y} : (\mathcal{X} \times \mathcal{Y})^* \times \mathcal{X}^* \to \mathcal{Y}^{\mathcal{X}}$ (called learning algorithm),
  where $\mathcal{Y}^{\mathcal{X}}$ is the space of all function from $\mathcal{X}$ to $\mathcal{Y}$

find a function

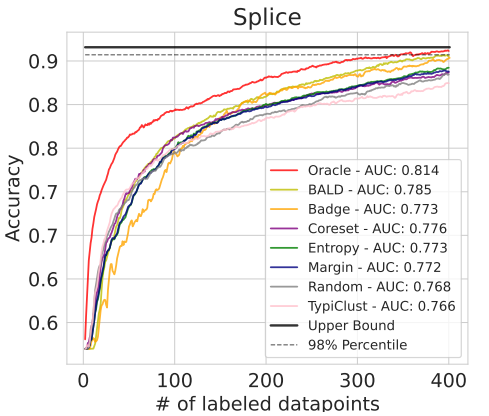
$$\Omega : (\mathcal{X} \times \mathcal{Y})^* \times \mathcal{X}^* \to \mathbb{N} \qquad \text{(with } a(\mathcal{D}, X) \leq |X|\text{)}$$

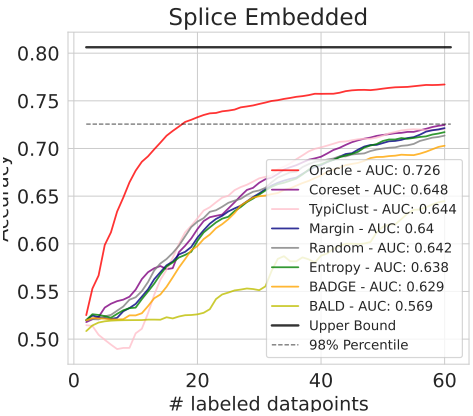
where $a(\mathcal{D}, X)$ selects an unlabeled instance from X

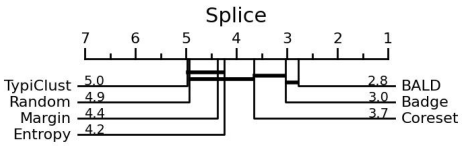
called acquisition function, s.t. the expected loss of a model learned on all predictors plus B sequentially acquired targets is minimal:

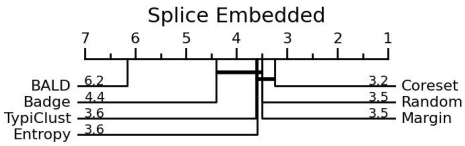
$$\begin{aligned} &\min\ \mathbb{E}\left\{\ell(\hat{y}, \mathcal{D}\text{test}) \mid \mathcal{D} \sim p, (\mathcal{D}\text{train}, \mathcal{D}\text{test}) := \text{split}(\mathcal{D})\right\} \\ &\text{with } \hat{y} := A((\mathcal{D}_{\text{train}_{n_1}}, \ldots, \mathcal{D}_{\text{train}_{n_B}}), \mathcal{D}\text{train}|_{\mathcal{X}})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

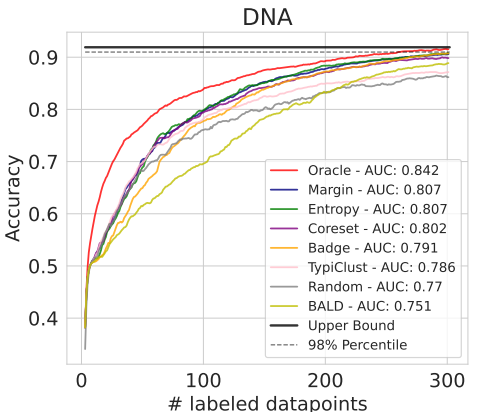
B All Results

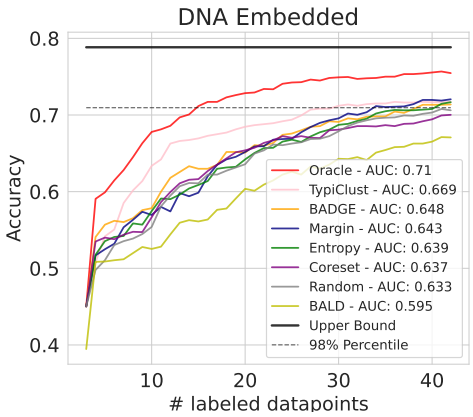


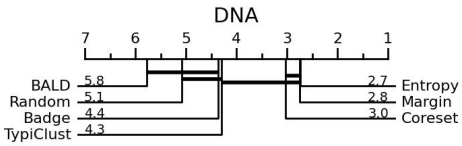


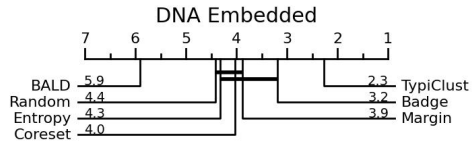


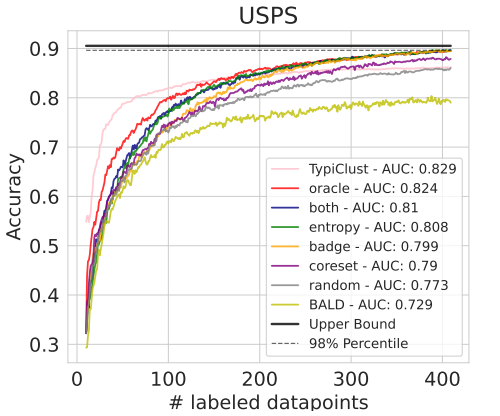


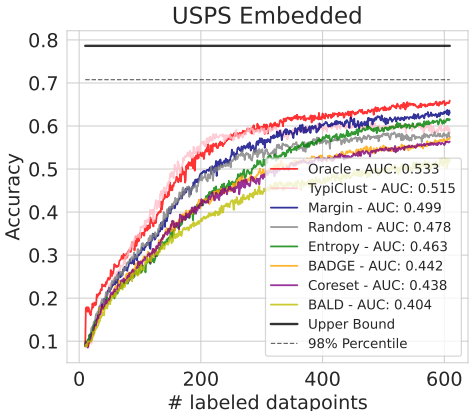


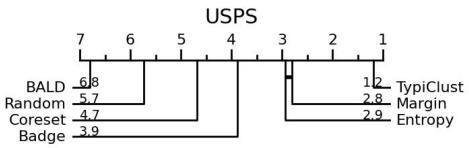


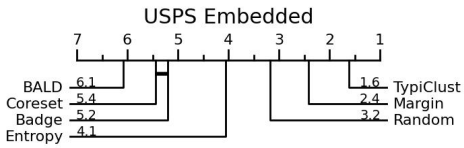


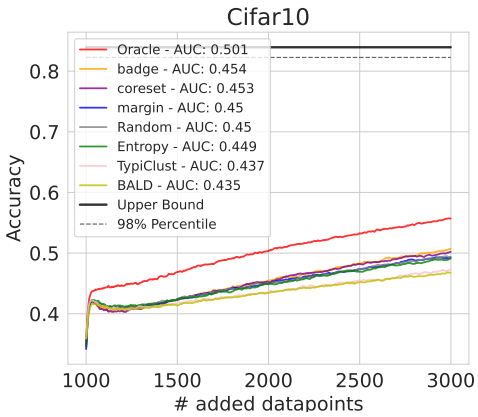


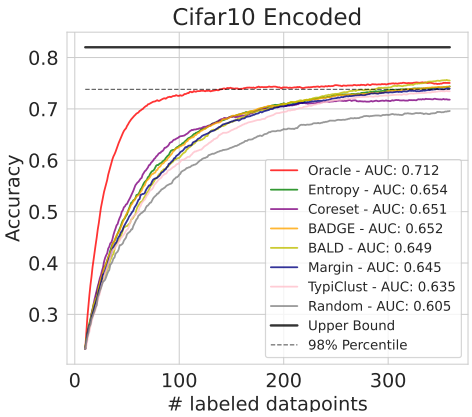

USPS
Accuracy
labeled datapoints
TypiClust - AUC: 0.829
oracle - AUC: 0.824
both - AUC: 0.81
entropy - AUC: 0.808
badge - AUC: 0.799
coreset - AUC: 0.79
random - AUC: 0.773
BALD - AUC: 0.729
Upper Bound
98% Percentile

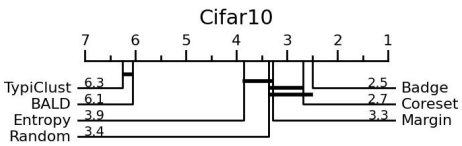

USPS Embedded
Accuracy
labeled datapoints
Oracle - AUC: 0.533
TypiClust - AUC: 0.515
Margin - AUC: 0.499
Random - AUC: 0.478
Entropy - AUC: 0.463
BADGE - AUC: 0.442
Coreset - AUC: 0.438
BALD - AUC: 0.404
Upper Bound
98% Percentile

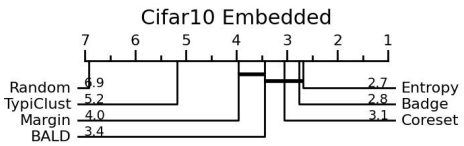

USPS
BALD 6.8
Random 5.7
Coreset 4.7
Badge 3.9
TypiClust 1.2
Margin 2.8
Entropy 2.9

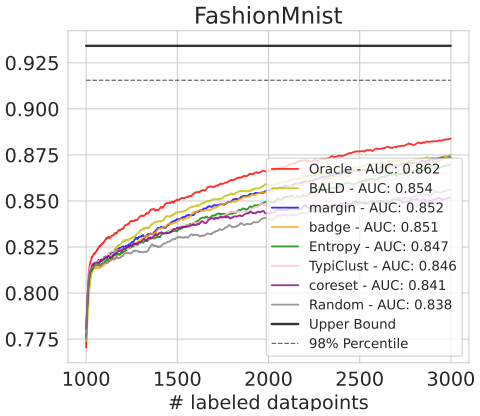

USPS Embedded
BALD 6.1
Coreset 5.4
Badge 5.2
Entropy 4.1
TypiClust 1.6
Margin 2.4
Random 3.2

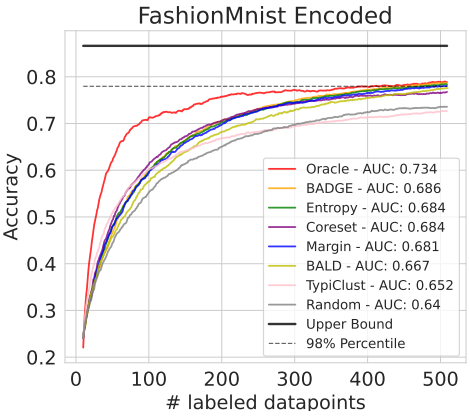

Cifar10
Accuracy
added datapoints
Oracle - AUC: 0.501
badge - AUC: 0.454
coreset - AUC: 0.453
margin - AUC: 0.45
Random - AUC: 0.45
Entropy - AUC: 0.449
TypiClust - AUC: 0.437
BALD - AUC: 0.435
Upper Bound
98% Percentile

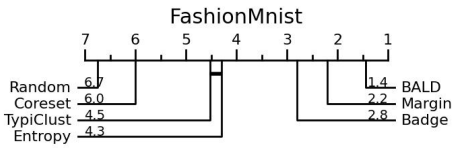

Cifar10 Encoded
Accuracy
labeled datapoints
Oracle - AUC: 0.712
Entropy - AUC: 0.654
Coreset - AUC: 0.651
BADGE - AUC: 0.652
BALD - AUC: 0.649
Margin - AUC: 0.645
TypiClust - AUC: 0.635
Random - AUC: 0.605
Upper Bound
98% Percentile

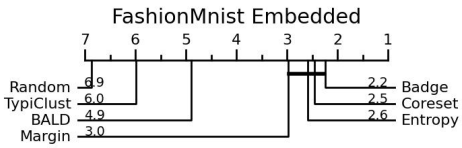

Cifar10
TypiClust 6.3
BALD 6.1
Entropy 3.9
Random 3.4
Badge 2.5
Coreset 2.7
Margin 3.3

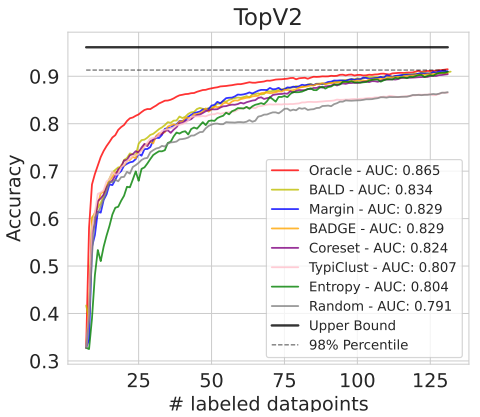

Cifar10 Embedded
Random 6.9
TypiClust 5.2
Margin 4.0
BALD 3.4
Entropy 2.7
Badge 2.8
Coreset 3.1

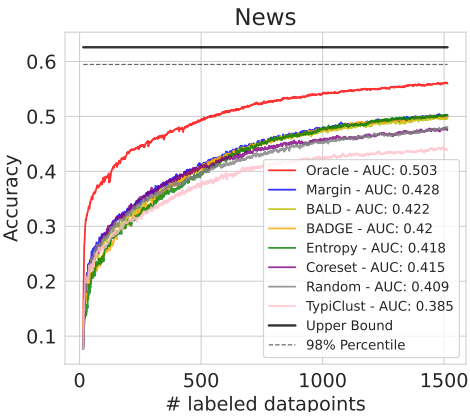

FashionMnist
Oracle - AUC: 0.862
BALD - AUC: 0.854
margin - AUC: 0.852
badge - AUC: 0.851
Entropy - AUC: 0.847
TypiClust - AUC: 0.846
coreset - AUC: 0.841
Random - AUC: 0.838
Upper Bound
98% Percentile
labeled datapoints

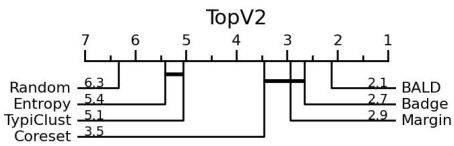

FashionMnist Encoded
Oracle - AUC: 0.734
BADGE - AUC: 0.686
Entropy - AUC: 0.684
Coreset - AUC: 0.684
Margin - AUC: 0.681
BALD - AUC: 0.667
TypiClust - AUC: 0.652
Random - AUC: 0.64
Upper Bound
98% Percentile
Accuracy
labeled datapoints

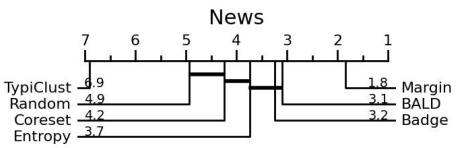

FashionMnist
Random 6.7
Coreset 6.0
TypiClust 4.5
Entropy 4.3
BALD 1.4
Margin 2.2
Badge 2.8


FashionMnist Embedded
Random 6.9
TypiClust 6.0
BALD 4.9
Margin 3.0
Badge 2.2
Coreset 2.5
Entropy 2.6

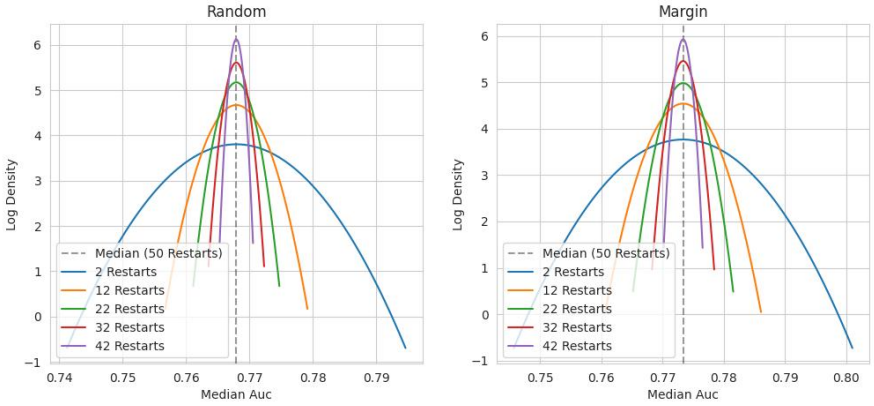

TopV2
Oracle - AUC: 0.865
BALD - AUC: 0.834
Margin - AUC: 0.829
BADGE - AUC: 0.829
Coreset - AUC: 0.824
TypiClust - AUC: 0.807
Entropy - AUC: 0.804
Random - AUC: 0.791
Upper Bound
98% Percentile
Accuracy
labeled datapoints


News
Oracle - AUC: 0.503
Margin - AUC: 0.428
BALD - AUC: 0.422
BADGE - AUC: 0.42
Entropy - AUC: 0.418
Coreset - AUC: 0.415
Random - AUC: 0.409
TypiClust - AUC: 0.385
Upper Bound
98% Percentile
Accuracy
labeled datapoints


TopV2
Random 6.3
Entropy 5.4
TypiClust 5.1
Coreset 3.5
BALD 2.1
Badge 2.7
Margin 2.9


News
TypiClust 6.9
Random 4.9
Coreset 4.2
Entropy 3.7
Margin 1.8
BALD 3.1
Badge 3.2

C Alternative Plot for Restarts Ablation



D Hyperparameters per AL Algorithm

TODO

E 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 3: Classifier architectures and optimized hyperparameters per dataset. Numbers in brackets signify a MLP with corresponding hidden layers.

F 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)

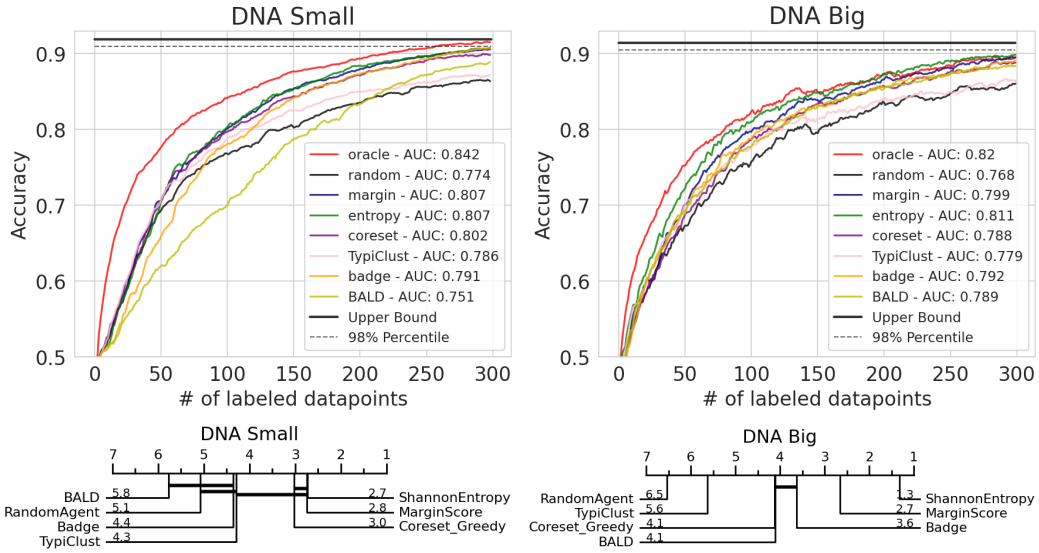


Figure 4: Comparison of small and big classifiers for the DNA dataset

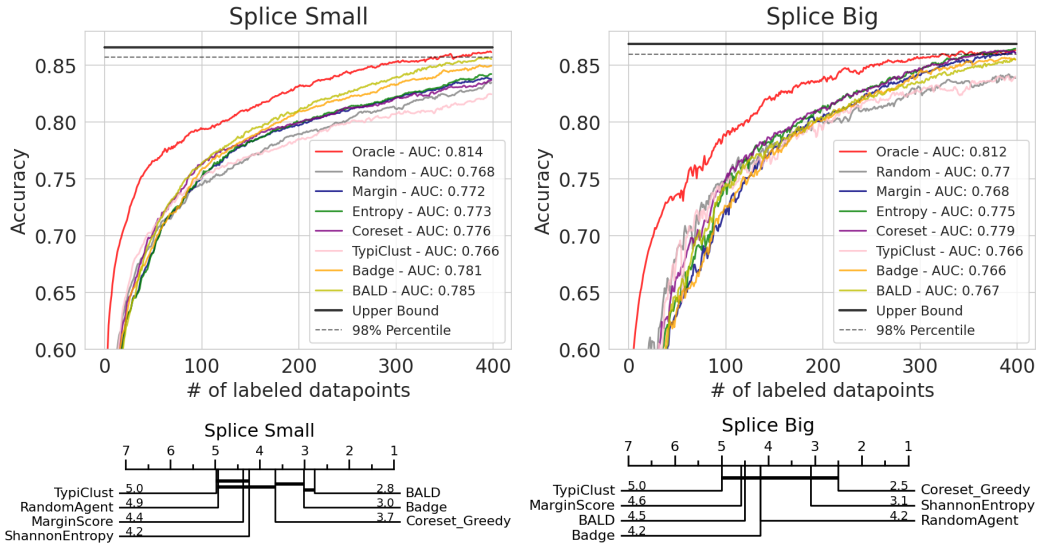
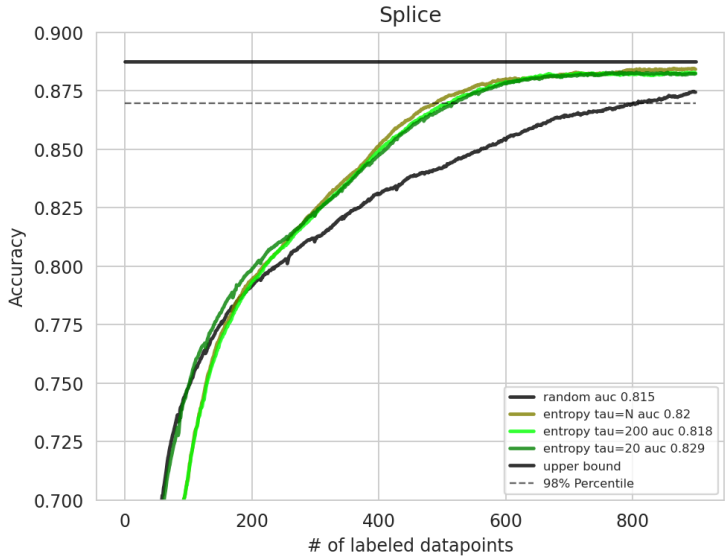


Figure 5: Comparison of small and big classifiers for the Splice dataset

G Comparison of different sample sizes



H AL Pseudocode

Algorithm 2 Active Learning

Require: $\mathcal{U}$ ▷ Unlabeled Pool
Require: τ ▷ Unlabeled Sample Size
Require: Ω ▷ AL Agent
Require: ω ▷ 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**
  $\text{acc}^{(i)} \leftarrow \text{Retrain}(\mathcal{L}^{(i)})$
  $a^{(i)} \leftarrow \Omega(\mathcal{U}^{(i)})\ ;\ \ a \in 1 : |\mathcal{U}^{(i)}|$
  $y^{(i)} \leftarrow \text{label}(\mathcal{U}_a^{(i)})$
  $\mathcal{L}^{(i+1)} \leftarrow \mathcal{L}^{(i)} \cup \{(\mathcal{U}_a^{(i)}, y^{(i)})\}$
  $\mathcal{U}^{(i+1)} \leftarrow \mathcal{U}^{(i)} \setminus \{\mathcal{U}_a^{(i)}\}$
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: $e^{\max}$ ▷ Maximum Epochs

$\text{loss}^* \leftarrow \infty$
for $i := 1 \dots e^{\max}$ **do**
    $\theta_{i+1} \leftarrow \theta_i - \eta \nabla_\theta \ell(\mathcal{L}, \hat{y}_\theta)$
    $\text{loss}_i \leftarrow \ell(\mathcal{D}^{\text{val}}, \hat{y}_\theta)$
    if $\text{loss}_i < \text{loss}^*$ **then**
        $\text{loss}^* \leftarrow \text{loss}_i$
    else
        Break
return $\text{Acc}(\mathcal{D}^{\text{test}}, \hat{y}_\theta)$