Towards Comparable Active Learning

Anonymous Authors

Abstract

Active Learning has received significant attention in the field of machine learning. Its goal is to select the most informative samples for labeling, thereby reducing data annotation costs. However, it has been brought to attention multiple times that reported lifts from literature generalize poorly and display high variance, leading to an inconclusive landscape in Active Learning research. Based on recent insights for reliable evaluation for Active Learning, this work extends experimentation from the commonly used image domain to a wide spectrum of domains. Additionally, we provide an analysis of how many repetitions an Active Learning experiment needs in order to derive conclusive results and propose that previous benchmarks have not met the necessary number of repetitions. To the best of our knowledge, we propose the first AL benchmark that applies state-of-the-art evaluation on algorithms in 3 major domains: Tabular, Image, and Text as well as synthetic data. We report empirical results for 11 widely used algorithms on 7 real-world and 2 synthetic datasets and aggregate them into domain-specific and overall rankings 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 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, 9, 15, 29].

On top of providing a principled way to label 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. In the last decade, many different algorithms for AL have been proposed. Even though, almost every method has reported lifts over all its predecessors, [1] AL research faces four central difficulties: (i) The experiments are often carried out on different datasets and model architectures, hindering direct comparison, (ii) generalize poorly across different domains, (iii) the reported results can be subject to very high variance across restarts and (iv) are not always compared against important baselines like margin sampling [24]. While multiple benchmark suites have been proposed to solve (i), to the best of our knowledge, we are the first to report results on all 3 data domains of tabular, image and text. Additionally, we provide synthetic datasets to highlight principled shortcoming of existing AL algorithms. Regarding (ii) and (iii), [29] 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

[1] Out of all considered algorithms for this paper, only BALD [7] did not claim a new SOTA performance in their result section.

Code available at: anonymous

Submitted to the 38th Conference on Neural Information Processing Systems (NeurIPS 2024) Track on Datasets and Benchmarks. Do not distribute.

Table 1: Comparison of our benchmark with the existing literature. Oracle curves serve as an approximation of the best possible AL algorithm. Including the encoded versions of our datasets we reach 14 datasets. "Semi" indicates whether the paper is employing any form of self- or semi-supervised learning. A "-" for repetitions means that we could not determine how often each experiment is repeated in the respective framework.

Paper	Sampling	#Data	#Alg	Img	Txt	Tab	Synth	Semi	Oracle	Repetitions
Beck et al. [2]	batch	4	7	✓	-	-	-	-	-	-
Hu et al. [9]	batch	5	13	✓	✓	-	-	-	-	3
Zhou et al. [29]	batch	3	2	✓	✓	-	-	-	✓	5
Zhan et al. [27]	sngl+batch	35	18	-	-	✓	✓	-	✓	10-100
Munjal et al. [19]	batch	2	8	✓	-	-	-	-	-	3
Li et al. [15]	batch	5	13	✓	-	-	-	✓	-	-
Rauch et al. [22]	batch	11	5	-	✓	-	-	-	-	5
Ji et al. [10]	batch	3	8	✓	-	-	-	-	-	-
Lueth et al. [17]	batch	4	5	✓	-	-	-	✓	-	3
Ours	sngl+batch	9(14)	11	✓	✓	✓	✓	✓	✓	50

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 ([29] Appendix A). These outlined issues lead to an inconclusive landscape of AL algorithms, where the vast majority of reported lifts are neither statistically significant, nor prove to be generalizable. This makes it very hard to identify the best AL algorithm, or even identifying state-of-the-art algorithms. In this work we propose an evaluation protocol that was designed to handle 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. We base our work largely on [10], following their guidelines for a reliable evaluation of AL algorithms, while extending the number of evaluated data domains from 1 to 5.

We focus on pool-based AL where a pool of unlabeled samples is fixed at the start of each experiment and one or more samples are chosen sequentially. In addition to the default scenario of selecting a batch of samples every iteration we incorporate the single sample case into our benchmark. Batched algorithms (and benchmarks) do not have a principled advantage over single-sample AL except for 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 leading to a generally worse performance, that impacts some algorithms more than others. We propose single-sample AL as an important tool to identify the best acquisition function, rather than the best combination of acquisition function and diversity regularization.

Table 1 shows a feature comparison between our proposed benchmark and several existing benchmarks in the literature. We offer our datasets in two versions - normal and encoded. An encoded dataset was pre-encoded by an self-supervised encoder model, providing a different use case for active learning. Our synthetic and text datasets (which use word embeddings in the normal setting) are exempt from this, bringing our total dataset count to 14 across 5 domains (Tabular, Image, Text, Synthetic, Encoded).

Contributions

- C1 Study on the reproducibility of AL experiments, providing evidence that previous works might have not repeated their experiments often enough to provide reliable results
- C2 Efficient and performant algorithm for an oracle that can be constructed greedily and does not rely on search
- C3 Two novel synthetic datasets named "Honeypot" and "Diverging Sine" that highlight two principled shortcomings AL algorithms. Firstly, a susceptibility to noisy or adverse samples and secondly, the oversampling of easy to distinguish regions of the dataset
- C4 Extending the evaluation of Active Learning algorithms to 5 data domains and revealing significant differences in algorithm performance between them

2 Problem Description

Given two spaces $\mathcal{X}, \mathcal{Y}$, $n = l + u$ data points with $l \in \mathbb{N}$ labeled examples $\mathcal{L} = \{(x_1, y_1), \dots, (x_l, y_l)\}$, $u \in \mathbb{N}$ unlabeled examples $\mathcal{U} = \{x_{l+1}, \dots, x_n\}$, a model $\hat{y} : \mathcal{X} \to \mathcal{Y}$, a budget $\mathbb{N} \ni b \leq u$ and an annotator $A : \mathcal{X} \to \mathcal{Y}$ 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 $\Omega : \mathcal{U}^{(i)}, \mathcal{L}^{(i)} \mapsto x^{(i)} \in \mathcal{U}^{(i)}$ that iteratively selects the next unlabeled point $x^{(i)}$ for labeling

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

with $\mathcal{U}^{(0)} = \mathrm{seed}(\mathcal{U}, s)$ and $\mathcal{L}^{(0)} = \left(\mathcal{U}_i^{(0)}, A(\mathcal{U}_i^{(0)})\right)\ i \in [1, \dots, s]$, where $\mathrm{seed}(\mathcal{U}, s)$ selects s points per class for the initial labeled set.
So that the average expected loss $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ of a machine learning algorithm fitting $\hat{y}^{(i)}$ on the respective labeled set $\mathcal{L}^{(i)}$ is minimal:

$$\min \quad \frac{1}{B} \sum_{i=0}^{B} \mathbb{E}_{(x,y)\sim\rho} \ell(y, \hat{y}^{(i)})$$

3 Related Work

While multiple benchmark suites have been proposed for Active Learning, none of them provide experiments for more than two domains. The authors of [2], [19], [15], [10] and [17] even focus exclusively on the image domain. Experiments on the interplay between AL and semi-supervised learning have only been provided by two works so far [15, 17], both of them only for images. An oracle algorithm has so far been proposed by only two works [29, 28]. Both of these algorithms rely on search, while our proposed method can be constructed sequentially. The two closest related works to this benchmark are [10] and [17], who also place a much higher emphasis on the problem of evaluating AL algorithms under many forms of variance than their predecessors (indicated in Tab. 1 by a dashed line). The authors of [10] posed a total of 12 ”recommendations” for reliable evaluation of AL algorithms. We largely adapt the proposed recommendations of [10] and extend their work to multiple domains, batch sizes and comparisons. For a complete list of the recommendations and our implementation of them, please refer to App. A. This work also pays attention to the so-called ”pitfalls” of AL evaluation proposed in [17]. For a complete list of the pitfalls and our implementation of them, please refer to App. B. To the best of our knowledge, we are the first to extend reliable SOTA (based on [10, 17]) experimentation to a total of 5 data domains.

4 Methodology

4.1 Why we need 50 restarts

To evaluate how many restarts are necessary to obtain conclusive results in an AL experiment, we computed 100 runs of our top-performing algorithm on one dataset. Our best algorithm is margin sampling and we chose the Splice dataset for its average size and complexity.
This allows us firstly, to obtain a very strong estimation of the ”true” average performance of margin sampling on this dataset and secondly, to draw subsets from this pool of 100 runs. Setting the size of our draws to α and sampling uniformly, we can approximate a cross-validation process with α restarts. Each of these draws can be interpreted as a **reported result in AL literature** where the authors employed α restarts. Figure 1 shows the ”true” mean performance of margin sampling (green) in relation to random sampling (black) and the oracle performance (red). We display 5 random draws of size α in blue. We can observe that even for a relatively high number of restarts the variance between the samples is extremely high, resulting in some performance curves being worse

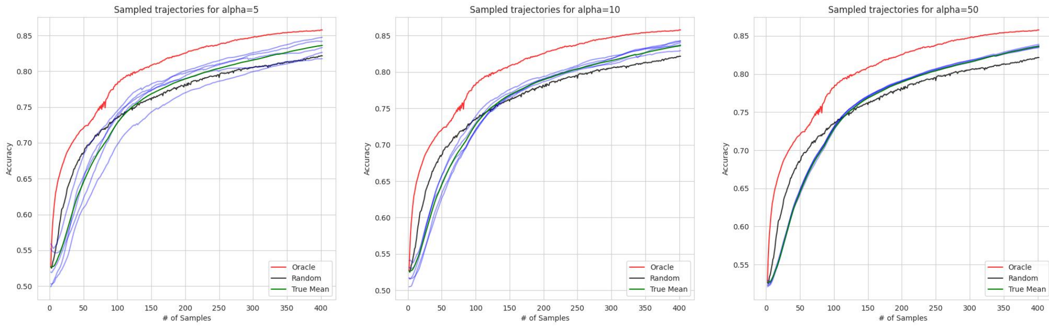


Figure 1: Random draws from a pool of 100 runs for margin sampling on the Splice dataset with different numbers of repetitions ($\alpha = \{5, 10, 50\}$). Green curves are the mean performance of all 100 runs, while the samples are blue. Even with 5 or 10 repetitions, we can observe that single draws for margin sampling display below-random performance (black), while the true mean should be above random.

that random and some being significantly better. When setting $\alpha = 50$ we observe all samples to converge close to the true mean performance. In addition to this motivating example, we carried out our main evaluation (Tab. 3) multiple times by uniformly sampling 3 random from our 50 available runs and comparing the results. We found significant differences in the performance of acquisition functions on individual datasets, as well as permutations in the final ranking. This partly explains the ongoing difficulties in reproducing results for AL experiments and benchmarks. This details can be found in App. D. For this benchmark we employ 50 restarts of every experiment.

4.2 Seeding vs. Restarts

Considering the high computational cost of 50 repetitions, another approach to ensure reproducibility would be to reduce the amount of variance in the experiment by keeping as many subsystems (weight initialization, data splits, etc.) as possible fixed with specialized seeding.
We describe a novel seeding strategy in Appendix H that creates 3 separate Random Number Generators (RNG) based on 3 different seeds. In short, 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, while $s_\mathcal{D}$ and s_θ are incremented by 1 between restarts to introduce stochasticity into our framework. While this seeding strategy is capable of controlling the amount variance in the experiment, previous works have noted that an actively sampled, labeled set does not generalize well between model architectures or even different initializations of the same model ([29, 16]), reducing its value in practice and providing a bad approximation of the quality of an AL algorithm. Hence, we opt for letting the subsystems vary (by increasing $s_\mathcal{D}$ and s_θ) and combine that with a high number of restarts to obtain a good average of the generalization performance of each AL algorithm.
Where a high number of restarts is computationally not feasible, we advise to additionally keep either $s_\mathcal{D}$ or s_θ (or both) fixed.

4.3 Datasets

A detailed description of the preprocessing of each dataset can be found in Appendix K.

Tabular: AL research conducted on tabular data is sparse (only [1] from the considered baseline papers). We, therefore, introduce a set of tabular datasets that we selected according to the following criteria: (i) They should be solvable by medium-sized models in under 1000 samples, (ii) the gap between most AL algorithms and random sampling should be significant (potential for AL is present) and (iii) the gap between the AL algorithms and our oracle should also be significant (research on

these datasets can produce further lifts). We use **Splice**, **DNA** and **USPS** from LibSVMTools [20].
Image: We use **FashionMNIST** [25] and **Cifar10** [13], since both are widely used in AL literature.
Text: We use **News Category** [18] and **TopV2** [6]. Text datasets have seen less attention in AL research, but most of the papers that evaluate on text ([9], [29]) use at least one of these datasets.

We would like to point out that these datasets are selected for speed of computation (both in terms of number of features and necessary budget to solve the dataset). However, similar to our argumentation for picking smaller classifiers, we are solely focused on comparing different AL algorithms in this paper and do not aim to develop novel classification models on these datasets. Our assumption is that a well-performing algorithm in our benchmark will also generalize well to larger real-world datasets, because we included multiple different data domains and classifier sizes in our experiments.

Adapting the experimental setting from [8], we offer all our datasets in the un-encoded (normal) setting as well as pre-encoded by a fixed embedding model that was trained by unsupervised contrastive learning. The text datasets are an exception to this, as they are only offered in their encoded form. The pre-encoded datasets enable us to test single-sample algorithms on more complex datasets like Cifar10 and FashionMnist. They also serve the purpose of investigating the interplay between self-supervised learning techniques and AL, as well as alleviating the cold-start problem described in [17] as they require a way smaller seed set. The classification model for every encoded dataset is a single linear layer with softmax activation. The embedding model was trained with the SimCLR [5] algorithm adopting the protocol from [8]. To ensure that enough information from the data is encoded by our embedding model, the quality of embeddings during pretext training was measured after each epoch. We attached a linear classification head to the encoder, fine-tuned it to the data and evaluated this classifier for test accuracy, mirroring our AL setup for embedded datasets. The checkpoint of each encoder model will be provided together with the framework.

Every dataset has a fixed size for the seed set of 1 sample per class, with the only exceptions being un-encoded FashionMnist and Cifar10 with 100 examples per class to alleviate the cold-start problem in these complex domains.

4.4 Batch Sizes

We selected batch sizes for each dataset to accommodate the widest range possible that results in a reasonable runtime for low batch sizes and allows for at least 4 round of data acquisition for high batch sizes. The available batch sizes per dataset can be found in Table 2.

Table 2: Employed model, chosen budget and available batch sizes for each dataset

	Model	B	1	5	20	50	100	500	1K
Enc. DNA	Linear	40	o	o					
Enc. Splice	Linear	100	o	o	o	o			
TopV2	BiLSTM	200	o	o	o	o			
Splice	MLP	400	o	o	o	o	o		
DNA	MLP	300	o	o	o	o	o		
USPS	MLP	400	o	o	o	o	o		
Enc. Cifar10	Linear	450	o	o	o	o	o		
Enc. FMnist	Linear	500	o	o	o	o	o		
Enc. USPS	Linear	600	o	o	o	o	o		
News	BiLSTM	3K			o	o	o	o	
FMnist	ResNet18	10K						o	o
Cifar10	ResNet18	10K						o	o

4.5 Realism vs. Variance

We would like to point out that some design choices for this framework prohibit direct transfer of our results to practical applications. This is a conscious choice, as we think that this is a necessary trade-off between realism and experiment variance. We would like to highlight the following design decisions:
(i) Creating test and validation splits from the full dataset rather than only the labeled seed set. Fully fledged test and validation splits are unobtainable in practice, but they provide not only a better approximation of algorithm performance, but also a better foundation for hyperparameter tuning, which is bound to reduce variance in the experiment.
(ii) Choosing smaller classifiers instead of SOTA models. Since we are not interested in archiving a new SOTA in any classification problem, we instead opt to use smaller classifiers for the following reasons: Smaller classifiers generally exhibit more stable training behavior, on average require fewer sampled datapoints to reach their full-dataset-performance and have faster training times. For every

dataset, the chosen architecture's hyperparameters are optimized to archive maximum full-dataset performance. Generally, we use MLPs for tabular, RestNet18 for image and BiLSTMs for text datasets. Every encoded dataset is classified by a single linear layer with softmax activation. The used model for each dataset can be found in Tab. 2. For a detailed description and employed hyperparameters please refer to Appendix K.

4.6 Greedy Oracle Algorithm

Posing Active Learning as a combinatorial problem, the oracle set $\mathcal{O}_b$ 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 computationally infeasible for realistic datasets, previous works have proposed approximations to this oracle sequence. [29] 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. Our oracle algorithm evaluates every data point $u_k = \text{unif}(\mathcal{U}) \quad \text{k} \in [1 \dots \tau]$ 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 over-specializing 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 classify 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. This leads to a situation, where no point can immediately incur an increase in test performance and therefore the selected data point can be considered random. To circumvent this problem, we use margin sampling [24] 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 ($\mathcal{O}(B\tau)$). We fix $\tau = 20$ in this work, as this gave us already a significant lift and we expect diminishing returns for larger τ. The pseudocode for our oracle can be found in App. L. Even though our proposed algorithm is more efficient than other approaches, the computational costs for high budget datasets like Cifa10 and FashionMnist meant that we could not compute the oracle for all 10000 datapoints. To still provide an oracle for these two datasets, we select two points per iteration instead of one and stop the oracle computation at a budget of 5000. The rest of the curve is forecast with a simple linear regression that asymptotically approaches the upper bound performance of the dataset. A detailed description can be found in App. I.

4.7 Evaluation Protocol

Following [29], 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):

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

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 AUC is still influenced by the budget, we define a set of rules to set this hyperparameter upfront, so that we are not favoring a subset of algorithms by handcrafting a budget. In this work, we choose the budget per dataset to be the first point at which one of 2 stopping conditions apply: (i) an algorithm (except Oracle) manages to reach 99% of the full-dataset-performance (using the smallest query size) or (ii) the best algorithm (except oracle) did not improve the classifier's accuracy by at least 2% in the last 20% of iterations. The first rule follows [10], while the second rule prevents excessive budgets for cases with diminishing returns in the budget. The resulting budgets can be found in Tab. 2.

As described in Sec. 4.1, we restart each experiment multiple times. Each restart retains the train/test split (often given by the dataset itself), but creates a new validation split that is sampled (based on $s_{\mathcal{D}}$) from the entire dataset (not just the seed set $\mathcal{L}^{(0)}$).

Apart from plotting standard performance curves and reporting their AUC values per dataset in App. G, we primarily rely on ranks to aggregate the performance of an acquisition function across datasets. For each dataset and query size, the AUC values of all acquisition functions are sorted and assigned a rank based on position, with the best rank being 1. These ranks can safely be averages across datasets as they are no longer subjected to scaling differences of each dataset. Additionally, we employ Critical Difference (CD) diagrams (like Fig. 2) for statistical testing. CD diagrams use the Wilcoxon signed-rank test, which is a variant of the paired T-test, to find significant differences of ranks between acquisition functions. For these diagrams, each combination of dataset, query size and run is considered a separate experiment, i.e. the results of `Dataset1-QuerySize1-run5` of an acquisition function `x` is only compared to the results of `Dataset1-QuerySize1-run5` of acquisition function `y`. Due to the large number of restarts and the wide range of datasets and query sizes, we can provide very accurate significance tests. For a detailed description of how every CD diagram is created, please refer to App. F.

5 Experiments

5.1 Implementation Details

At each iteration i the acquisition function Ω picks an unlabeled datapoint based on a fixed set of information $\{\mathcal{L}^{(i)}, \mathcal{U}^{(i)}, B, |\mathcal{L}^{(i)}| - |\mathcal{L}^{(1)}|, \mathrm{acc}^{(i)}, \mathrm{acc}^{(1)}, \hat{y}^{(i)}, \mathrm{opt}_{\hat{y}}\}$, where $\mathrm{opt}_{\hat{y}}$ is the optimizer used to fit $\hat{y}^{(i)}$. This set grants full access to the labeled and unlabeled set, as well as all parameters of the classifier and the optimizer. Additionally, we provide meta-information, like the size of the seed set through $|\mathcal{L}^{(i)}| - |\mathcal{L}^{(1)}|$, the remaining budget though the addition of B and the classifiers potential though $\mathrm{acc}^{(1)}$ and $\mathrm{acc}^{(i)}$. We allow acquisition functions to derive information from this set, e.g. predictions of the classifier $\hat{y}^{(i)}(x); \;\; x \in \mathcal{U}^{(i)} \cup \mathcal{L}^{(i)}$, clustering, or even training additional models. However, the algorithm may not incorporate external information e.g. other datasets, queries to recover additional labels, additional training steps for $\hat{y}$, or the test/validation set.

For our study we selected acquisition functions with good performances reported by multiple different sources that can work with the set of information stated above. For a list of all acquisition functions, please refer to Table 3, with detailed descriptions being found in Appendix C.

The model $\hat{y}$ can be trained in two ways. Either the parameters of the model are reset to a fixed initial setting $\hat{y}^{(0)}$ after each AL iteration and the classifier is trained from scratch with the updated labeled set $\mathcal{L}^{(i)}$, or the previous state $\hat{y}^{(i-1)}$ is retained and the classifier is fine-tuned on $\mathcal{L}^{(i)}$ for a reduced number of epochs. In this work, we use the fine-tuning method for un-encoded datasets to save computational time, while we use the from-scratch training for embedded datasets since they have very small classifiers and this approach generally produces better results. Our fine-tuning scheme always trains for at least one epoch and employs an aggressive early stopping with a patience of 2 afterwards.

5.2 Results on Real-world Data

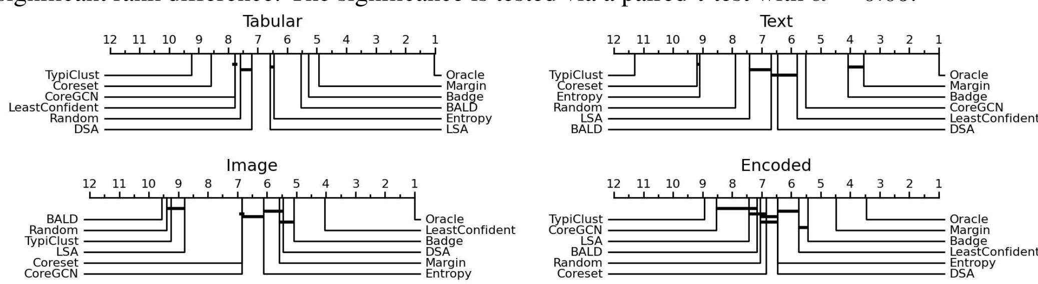
In Table 3 we provide the rank of each acquisition function per dataset and averaged for each (un-)encoded dataset. Please note, that for Tab 3 we are averaging not only over runs, but also over query sizes per dataset. For the results per query size, please refer to App. E.

As stated in contribution C4, our results on real-world data shows significant differences in the performance of the tested algorithms between data domains. Not only do some algorithms overperform on some domains (like least confidence sampling on Images), but the Top-3 of algorithms (except

Table 3: Performances for acquisition functions on real-world datasets, aggregated for un-encoded and encoded datasets. Performance is shown as average ranks over restarts (1.0 is the best rank). Algorithms are sorted by aggregated performance on un-encoded datasets.

	Splice	DNA	USPS	Cfr10	FMnist	TopV2	News	Un-enc.	Enc.
Oracle	1.0 ± 0.01	1.0 ± 0.01	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.0	1.0 ± 0.01	1.0 ± 0.0	1.0	2.0
Margin	6.6 ± 0.02	4.3 ± 0.01	2.1 ± 0.01	6.3 ± 0.01	4.4 ± 0.0	2.4 ± 0.01	3.7 ± 0.0	4.3	4.2
Badge	5.2 ± 0.01	6.3 ± 0.01	2.9 ± 0.01	5.2 ± 0.01	4.7 ± 0.0	3.3 ± 0.01	3.5 ± 0.0	4.5	5.4
LeastConf	9.2 ± 0.02	10.3 ± 0.02	8.1 ± 0.02	2.1 ± 0.01	4.0 ± 0.0	7.9 ± 0.02	3.0 ± 0.01	6.4	6.5
DSA	7.4 ± 0.02	7.3 ± 0.01	7.5 ± 0.01	5.4 ± 0.01	5.1 ± 0.0	6.0 ± 0.02	7.3 ± 0.01	6.6	6.7
BALD	4.0 ± 0.01	4.7 ± 0.01	5.4 ± 0.01	12.0 ± 0.01	7.6 ± 0.0	7.6 ± 0.02	5.0 ± 0.0	6.6	7.6
CoreGCN	6.9 ± 0.01	4.9 ± 0.01	10.4 ± 0.01	7.6 ± 0.01	6.5 ± 0.01	4.0 ± 0.01	6.8 ± 0.0	6.7	8.2
Entropy	6.6 ± 0.02	3.9 ± 0.01	7.6 ± 0.01	7.6 ± 0.01	4.9 ± 0.01	9.8 ± 0.02	9.6 ± 0.0	7.1	6.5
LSA	6.1 ± 0.01	6.8 ± 0.01	5.3 ± 0.01	7.7 ± 0.01	10.6 ± 0.01	7.5 ± 0.01	7.3 ± 0.01	7.3	7.5
Random	9.0 ± 0.01	9.3 ± 0.01	5.3 ± 0.01	8.4 ± 0.01	11.1 ± 0.0	7.9 ± 0.01	8.0 ± 0.0	8.4	6.9
Coreset	7.1 ± 0.01	9.0 ± 0.01	10.5 ± 0.01	6.8 ± 0.01	7.1 ± 0.0	8.5 ± 0.02	10.8 ± 0.01	8.5	7.2
TypiClust	8.8 ± 0.01	10.2 ± 0.01	12.0 ± 0.02	7.9 ± 0.01	11.0 ± 0.01	12.0 ± 0.02	12.0 ± 0.01	10.5	9.2

Figure 2: Ranks of each acquisition function aggregated by domain. Horizontal bars indicate a **non**-significant rank difference. The significance is tested via a paired-t-test with $\alpha = 0.05$.



Oracle) does not contain the same three algorithms for any two domains. Most interestingly, the image domain, which received the most attention in benchmarking so far could even be considered an outlier, as this is the only domain where the Top-1 algorithm changes. This highlights the dire need for diverse data domains in AL benchmarking.

6 Synthetic Datasets for AL

AL approaches can be categorized into two types, uncertainty and geometric approaches. Typical members of the first category are variants of uncertainty sampling like entropy-, margin and least-confident-sampling [24] as well as BALD [7]. Typical members of the second category are clustering approaches like Coreset [23], BADGE [1] and TypiClust [8]. Both types of algorithms have principled shortcomings in terms of the utilized information that makes them unsuitable for certain data distributions. To test for these specific shortcomings, we created two synthetic datasets, namely "Honeypot" and "Diverging Sine", that are hard to solve for methods focused on the classifier's decision boundary or data clustering respectively. To avoid algorithms memorizing these datasets they are generated from scratch for each experiment, depending on $s_{\mathcal{D}}$.

Honeypot creates to two easy to distinguish clusters with 150 samples each and one overlapping "honeypot" that represents a noisy region of the dataset with potentially miss-labeled, miss-measured or generally adverse samples. This honeypot contains 150 samples of each class, creating a balance of 50% beneficial samples and 50% adverse samples in the dataset. The honeypot is located on the likely decision boundary of a classifier that is trained on the beneficial samples to maximize its negative impact on purely uncertainty based acquisition functions. Diverging Sine samples the datapoints for each class from two diverging sinusoidal functions that are originating from the same y-intercept. This creates a challenging region one the left hand side, where a lot of datapoints need to be sampled and an easy region on the right hand side, where very few datapoints are enough. The repeating nature of a sin function encourages diversity based acquisition functions to equally sample the entire length, drastically oversampling the right hand side of the dataset. Each

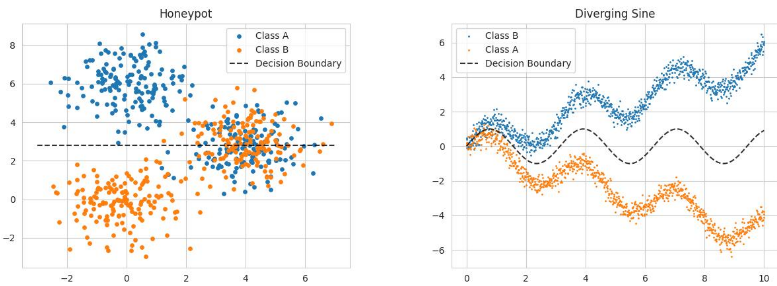
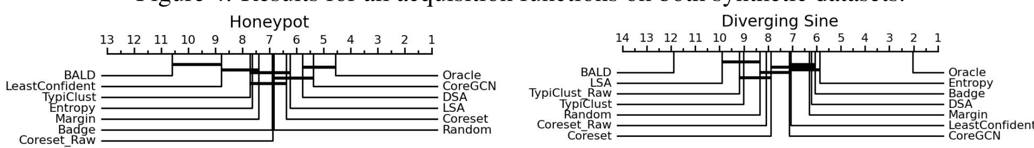


Figure 3: Synthetic ”Honeypot” and ”Diverging Sine” datasets. The optimal decision boundary is not part of the dataset and serves only as a visual guide.

Figure 4: Results for all acquisition functions on both synthetic datasets.



class has 500 datapoints. Both datasets have a budget of $B = 60$ and are tested with query sizes 1 and 5.

Results for the Honeypot dataset reveal expected shortcomings of uncertainty sampling algorithms like margin, entropy and least confident sampling as well as BALD. In addition, BADGE is under-performing for this dataset compared to real-world data. Results for Diverging Sine also confirm expected behavior, as clustering algorithms (Coreset, TypiClust) fall behind uncertainty algorithms (Entropy-, Margin-Sampling), with the exception of BADGE.

We provide a very small ablation study on the importance of the embeddings by testing a version of Coreset and TypiClust on this dataset that does not use the embeddings produced by the classification model, but rather clusters the data directly. ”Coreset Raw” and ”TypiClust Raw” both perform worse than their embedding-based counterpart.

6.1 Results on Synthetic Data

Our results on Honeypot reveal principled shortcomings for the two best algorithms in BADGE and margin sampling. Both are vulnerable to adverse samples or simply measurement noise, which highlights the need for further research in this area.

Finally, the fact that BADGE is able to perform well on Diverging Sine highlights the importance of embeddings for the clustering algorithms, as the so-called gradient embedding from BADGE seems to be able to encode uncertainty information, guiding the selection into the left hand regions of the dataset. We also show that embeddings are generally useful for this dataset, by providing results for ”Coreset Raw” and ”TypiClust Raw”.

7 Conclusion

We strongly advocate to test newly proposed AL algorithms not only on a wide variety of real data domains, but also to pay close attention to the Honeypot and Diverging Sine datasets to reveal principled shortcomings of the algorithm in question. Both tasks can be easily carried out by implementing the new acquisition function into our code base.

Acknowledgement anonymous

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] Razvan Caramalau, Binod Bhattarai, and Tae-Kyun Kim. Sequential graph convolutional network for active learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9583–9592, 2021.

[4] Akshay L Chandra. Deep active learning toolkit for image classification in pytorch. `https://github.com/acl21/deep-active-learning-pytorch`, 2021.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] Yilin Ji, Daniel Kaestner, Oliver Wirth, and Christian Wressnegger. Randomness is the root of all evil: More reliable evaluation of deep active learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3943–3952, 2023.

[11] Jinhan Kim, Robert Feldt, and Shin Yoo. Guiding deep learning system testing using surprise adequacy. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1039–1049. IEEE, 2019.

[12] 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.

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

[14] CURE Lab. Deep active learning with pytorch. `https://github.com/cure-lab/deep-active-learning`, 2022.

[15] 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.

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

[17] Carsten Lüth, Till Bungert, Lukas Klein, and Paul Jaeger. Navigating the pitfalls of active learning evaluation: A systematic framework for meaningful performance assessment. *Advances in Neural Information Processing Systems*, 36, 2024.

[18] Rishabh Misra. News category dataset. *arXiv preprint arXiv:2209.11429*, 2022.

[19] Prateek Munjal, Nasir Hayat, Munawar Hayat, Jamshid Sourati, and Shadab Khan. Towards robust and reproducible active learning using neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 223–232, 2022.

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

[21] 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.

[22] Lukas Rauch, Matthias Aßenmacher, Denis Huseljic, Moritz Wirth, Bernd Bischl, and Bernhard Sick. Activeglae: A benchmark for deep active learning with transformers. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 55–74. Springer, 2023.

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

[24] 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.

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

[26] Donggeun Yoo and In So Kweon. Learning loss for active learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 93–102, 2019.

[27] Xueying Zhan, Huan Liu, Qing Li, and Antoni B Chan. A comparative survey: Benchmarking for pool-based active learning. In *IJCAI*, pages 4679–4686, 2021.

[28] Xueying Zhan, Qingzhong Wang, Kuan-hao Huang, Haoyi Xiong, Dejing Dou, and Antoni B Chan. A comparative survey of deep active learning. *arXiv preprint arXiv:2203.13450*, 2022.

[29] 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 AL Recommendations from Ji et al.

TODO

B AL Pitfalls from Lueth et al.

TODO

C Acquistion Functions

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 [12] 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 the expected improvement in model performance.
Coreset [23] employs K-Means clustering trying 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 [8] relies on clustering similar to Coreset, but proposes a new measure called "Typicality" to select unlabeled samples. It 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 higher budgets.
Core-GCN [3] TODO
DSA/LSA [11] TODO

Excluded Algorithms
Learning Loss for AL [26] 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.

Reinforcement Learning Algorithms

D Difference of Ranks with 3 Repetitions

Table 4 and Table 5 follow the exact same computation of ranks that created the main result (Table 3) with the only difference being a reduced number of runs per acquisition function. For each table we uniformly sampled 3 runs from the available 50 per acquisition function.
We can observe significant differences between the two tables:
Purple: A multitude of rank differences of acquisition functions for specific datasets, some as high as 4.7 ranks for TypiClust on the Splice dataset
Olive: Well separated acquisition functions in Tab. 5 (Margin and BADGE) are almost indistinguishable in Tab 4
Red: BALD lost 2 places in the overall ranking and Entropy gained 2

Even though the overall ordering of acquisition functions stayed relatively unchanged due to the averaging across many datasets, each individual dataset was subject to drastic permutations. This highlights the need for many repetitions in AL experiments.

Table 4: Ranks of all acquisition functions per dataset. First random draw of 3 runs from the overall pool of 50.

	Splice	DNA	USPS	Cifar10	FMnist	TopV2	News	Unencoded	Encoded
Oracle	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.1
Margin	6.0	7.3	2.0	6.7	5.3	2.3	3.3	4.7	4.4
Badge	6.0	7.3	3.0	6.7	5.0	3.3	4.0	5.0	5.3
BALD	3.3	4.7	5.3	12.0	7.0	6.3	4.3	6.1	7.9
CoreGCN	8.7	3.7	10.7	6.3	5.3	4.0	7.7	6.6	9.1
DSA	8.3	6.3	7.7	7.7	4.3	6.7	6.7	6.8	6.1
LeastConf	10.0	12.0	8.0	3.0	4.3	9.3	2.3	7.0	6.7
LSA	5.7	6.7	5.3	6.7	10.7	7.7	7.0	7.1	6.3
Entropy	11.0	3.3	7.3	4.0	6.7	8.3	9.7	7.2	7.0
Random	7.7	8.7	5.3	8.0	11.0	8.0	9.0	8.2	6.3
Coreset	4.7	10.3	10.3	7.7	6.0	9.0	11.0	8.4	7.2
TypiClust	5.7	6.7	12.0	8.3	11.3	12.0	12.0	9.7	9.7

Table 5: Ranks of all acquisition functions per dataset. Second random draw of 3 runs from the overall pool of 50.

	Splice	DNA	USPS	Cifar10	FMnist	TopV2	News	Unencoded	Encoded
Oracle	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.4
Margin	6.0	3.3	2.0	5.7	2.0	2.0	4.3	3.6	3.8
Badge	6.0	9.0	3.0	3.0	5.7	3.7	3.3	4.8	4.9
CoreGCN	4.3	6.3	10.3	7.3	5.3	5.7	5.3	6.4	8.1
DSA	8.7	7.3	7.3	6.0	4.3	5.3	6.0	6.4	6.5
BALD	4.7	4.0	4.7	12.0	7.3	6.7	6.7	6.6	7.5
Entropy	6.7	4.7	7.7	5.3	5.0	7.3	9.3	6.6	6.8
LeastConf	7.7	10.0	8.3	3.3	6.0	8.7	3.0	6.7	7.3
LSA	7.7	5.3	6.0	9.0	11.0	9.0	7.3	7.9	7.5
Random	9.3	8.0	5.0	8.7	11.7	8.3	8.7	8.5	7.6
Coreset	6.0	10.7	10.7	8.0	8.3	8.3	11.0	9.0	6.3
TypiClust	10.0	8.3	12.0	8.7	10.3	12.0	12.0	10.5	9.4

E AUCs by Query Size

Table 6: AUC values for each dataset that supports query size 1.

	Splice	SpliceEncoded	DNA	DNAEncoded	USPS	USPSEncoded	Cifar10Encoded	FashionMnistEnc	TopV2	DivergingSin	ThreeClust
Oracle	0.803+-0.012	0.678+-0.021	0.825+-0.009	0.721+-0.013	0.866+-0.004	0.436+-0.057	0.749+-0.009	0.755+-0.005	0.884+-0.006	0.957+-0.009	0.783+-0.03
Margin	0.769+-0.021	**0.678+-0.032**	0.806+-0.013	0.642+-0.047	**0.858+-0.006**	0.426+-0.038	0.653+-0.013	0.68+-0.012	**0.861+-0.009**	0.941+-0.018	0.704+-0.074
Badge	0.767+-0.02	0.661+-0.026	0.78+-0.014	0.642+-0.046	0.83+-0.008	0.371+-0.035	0.656+-0.013	0.68+-0.009	0.826+-0.024	0.941+-0.017	0.69+-0.083
LeastConfident	0.779+-0.019	0.68+-0.032	0.809+-0.01	0.629+-0.05	0.846+-0.009	0.421+-0.039	**0.668+-0.014**	**0.685+-0.009**	0.843+-0.013	0.94+-0.016	0.692+-0.094
DSA	0.766+-0.021	0.691+-0.022	0.803+-0.01	**0.646+-0.032**	0.829+-0.01	**0.431+-0.05**	0.663+-0.014	0.679+-0.01	0.844+-0.017	0.941+-0.014	**0.731+-0.032**
BALD	**0.78+-0.014**	0.649+-0.04	0.784+-0.01	0.632+-0.042	0.819+-0.01	0.242+-0.046	0.666+-0.014	0.644+-0.018	0.815+-0.024	0.928+-0.014	0.698+-0.043
CoreGCN	0.765+-0.021	0.686+-0.023	0.804+-0.012	**0.646+-0.03**	0.753+-0.016	0.39+-0.044	0.623+-0.018	0.647+-0.012	0.85+-0.01	0.938+-0.014	**0.731+-0.028**
Entropy	0.768+-0.022	**0.678+-0.035**	**0.812+-0.013**	0.635+-0.045	0.83+-0.011	0.399+-0.035	0.663+-0.013	0.681+-0.011	0.815+-0.021	**0.942+-0.017**	0.696+-0.083
LSA	0.772+-0.016	0.68+-0.026	0.787+-0.012	0.618+-0.036	0.821+-0.009	0.422+-0.037	0.613+-0.014	0.642+-0.012	0.816+-0.013	0.932+-0.016	0.727+-0.033
Random	0.76+-0.016	0.674+-0.027	0.774+-0.013	0.63+-0.035	0.823+-0.009	0.404+-0.036	0.613+-0.014	0.639+-0.013	0.815+-0.012	0.933+-0.017	0.721+-0.036
Coreset	0.772+-0.016	0.69+-0.017	0.79+-0.012	0.638+-0.041	0.767+-0.016	0.404+-0.046	0.659+-0.011	0.684+-0.009	0.826+-0.022	0.937+-0.014	0.73+-0.031
TypiClust	0.762+-0.016	0.685+-0.025	0.778+-0.01	0.663+-0.028	0.828+-0.007	0.396+-0.046	0.653+-0.013	0.649+-0.007	0.831+-0.011	0.934+-0.018	0.727+-0.033

Table 7: AUC values for each dataset that supports query size 5.

	Splice	SpliceEncoded	DNA	DNAEncoded	USPS	USPSEncoded	Cifar10Encoded	FashionMnistEncoded	TopV2	DivergingSin	ThreeClust
Oracle	0.803+-0.012	0.678+-0.021	0.825+-0.009	0.721+-0.013	0.866+-0.004	0.436+-0.057	0.749+-0.009	0.755+-0.005	0.884+-0.006	0.957+-0.009	0.783+-0.03
Margin	0.765+-0.021	0.662+-0.032	0.794+-0.011	0.611+-0.05	0.855+-0.006	0.508+-0.02	0.656+-0.014	0.678+-0.009	0.848+-0.013	0.923+-0.019	0.697+-0.055
Badge	0.768+-0.014	0.646+-0.035	0.785+-0.011	0.624+-0.036	0.846+-0.007	0.48+-0.021	0.647+-0.012	0.67+-0.009	0.847+-0.01	0.924+-0.019	0.72+-0.036
LeastConfident	0.763+-0.023	0.643+-0.034	0.798+-0.013	0.585+-0.065	0.831+-0.014	0.478+-0.028	0.67+-0.01	0.681+-0.009	0.819+-0.023	0.921+-0.019	0.675+-0.072
DSA	0.765+-0.023	0.653+-0.029	0.793+-0.009	0.613+-0.034	0.822+-0.01	0.489+-0.024	0.661+-0.013	0.662+-0.012	0.833+-0.02	0.924+-0.018	0.718+-0.033
BALD	0.775+-0.018	0.641+-0.034	0.801+-0.013	0.592+-0.054	0.84+-0.008	0.332+-0.054	0.681+-0.011	0.681+-0.013	0.824+-0.023	0.893+-0.035	0.673+-0.041
CoreGCN	0.759+-0.018	0.662+-0.027	0.79+-0.011	0.62+-0.03	0.755+-0.011	0.45+-0.03	0.604+-0.016	0.609+-0.013	0.837+-0.014	0.922+-0.018	0.723+-0.034
Entropy	0.765+-0.022	0.66+-0.03	0.798+-0.011	0.611+-0.054	0.823+-0.013	0.464+-0.024	0.663+-0.013	0.672+-0.011	0.801+-0.025	0.924+-0.02	0.689+-0.066
LSA	0.769+-0.016	0.654+-0.032	0.781+-0.013	0.61+-0.041	0.82+-0.009	0.484+-0.022	0.617+-0.012	0.641+-0.011	0.816+-0.012	0.915+-0.018	0.718+-0.038
Random	0.758+-0.015	0.655+-0.026	0.771+-0.013	0.623+-0.031	0.82+-0.009	0.476+-0.024	0.616+-0.016	0.637+-0.012	0.812+-0.014	0.921+-0.018	0.713+-0.034
Coreset	0.765+-0.017	0.663+-0.023	0.784+-0.014	0.603+-0.034	0.765+-0.015	0.449+-0.022	0.657+-0.009	0.674+-0.009	0.817+-0.017	0.92+-0.017	0.713+-0.035
TypiClust	0.759+-0.014	0.641+-0.028	0.775+-0.01	0.603+-0.04	0.757+-0.02	0.465+-0.027	0.596+-0.014	0.567+-0.012	0.727+-0.026	0.916+-0.02	0.693+-0.045

Table 8: AUC values for each dataset that supports query size 20.

	Splice	SpliceEncoded	DNA	USPS	USPSEncoded	Cifar10Encoded	FashionMnistEnc	TopV2	News
Oracle	0.803+-0.012	0.678+-0.021	0.825+-0.009	0.866+-0.004	0.436+-0.057	0.749+-0.009	0.755+-0.005	0.884+-0.006	0.49+-0.003
Margin	0.759+-0.027	0.618+-0.04	0.779+-0.013	0.847+-0.008	0.439+-0.027	0.656+-0.01	0.67+-0.011	0.823+-0.014	0.464+-0.007
Badge	0.767+-0.013	0.619+-0.033	0.776+-0.013	0.845+-0.006	0.44+-0.019	0.647+-0.013	0.665+-0.007	0.827+-0.016	0.463+-0.007
LeastConfident	0.751+-0.02	0.597+-0.05	0.748+-0.025	0.798+-0.027	0.391+-0.024	0.665+-0.013	0.669+-0.011	0.775+-0.035	0.467+-0.008
DSA	0.759+-0.02	0.599+-0.034	0.769+-0.013	0.809+-0.012	0.421+-0.023	0.647+-0.014	0.63+-0.013	0.793+-0.026	0.459+-0.01
BALD	0.768+-0.022	0.57+-0.037	0.784+-0.015	0.822+-0.009	0.298+-0.039	0.675+-0.008	0.673+-0.01	0.789+-0.024	0.468+-0.009
CoreGCN	0.759+-0.018	0.612+-0.039	0.774+-0.012	0.754+-0.016	0.397+-0.026	0.587+-0.015	0.583+-0.015	0.807+-0.018	0.453+-0.006
Entropy	0.759+-0.027	0.618+-0.038	0.773+-0.015	0.803+-0.019	0.372+-0.022	0.656+-0.011	0.65+-0.012	0.773+-0.031	0.451+-0.007
LSA	0.761+-0.014	0.611+-0.039	0.768+-0.015	0.816+-0.009	0.411+-0.022	0.621+-0.01	0.635+-0.011	0.796+-0.016	0.452+-0.007
Random	0.755+-0.014	0.612+-0.039	0.763+-0.012	0.818+-0.009	0.439+-0.019	0.622+-0.013	0.633+-0.012	0.795+-0.016	0.45+-0.006
Coreset	0.759+-0.016	0.601+-0.034	0.764+-0.015	0.757+-0.015	0.39+-0.029	0.647+-0.009	0.651+-0.011	0.784+-0.026	0.435+-0.012
TypiClust	0.751+-0.012	0.551+-0.036	0.76+-0.016	0.643+-0.026	0.411+-0.024	0.488+-0.02	0.449+-0.017	0.652+-0.035	0.406+-0.011

Table 9: AUC values for each dataset that supports query size 50.

	Splice	DNA	USPS	USPSEncoded	Cifar10Encoded	FashionMnistEnc	TopV2	News
Oracle	0.803+-0.012	0.825+-0.009	0.866+-0.004	0.436+-0.057	0.749+-0.009	0.755+-0.005	0.884+-0.006	0.49+-0.003
Margin	0.747+-0.023	0.751+-0.019	0.828+-0.009	0.363+-0.031	0.64+-0.013	0.653+-0.01	0.774+-0.029	0.46+-0.006
Badge	0.758+-0.017	0.754+-0.018	0.831+-0.008	0.376+-0.028	0.632+-0.013	0.649+-0.011	0.781+-0.026	0.462+-0.007
LeastConfident	0.731+-0.025	0.688+-0.041	0.761+-0.037	0.291+-0.03	0.644+-0.013	0.65+-0.011	0.73+-0.049	0.462+-0.009
DSA	0.748+-0.021	0.738+-0.018	0.783+-0.016	0.346+-0.027	0.624+-0.014	0.588+-0.016	0.748+-0.041	0.45+-0.011
BALD	0.76+-0.017	0.756+-0.018	0.796+-0.016	0.241+-0.026	0.65+-0.009	0.645+-0.01	0.746+-0.038	0.455+-0.007
CoreGCN	0.755+-0.016	0.745+-0.018	0.752+-0.019	0.328+-0.027	0.581+-0.015	0.568+-0.018	0.771+-0.025	0.453+-0.007
Entropy	0.747+-0.024	0.748+-0.018	0.778+-0.024	0.275+-0.026	0.633+-0.011	0.625+-0.012	0.734+-0.036	0.442+-0.007
LSA	0.754+-0.013	0.749+-0.019	0.807+-0.01	0.341+-0.029	0.613+-0.012	0.625+-0.01	0.763+-0.025	0.45+-0.006
Random	0.746+-0.012	0.745+-0.015	0.806+-0.008	0.379+-0.028	0.615+-0.014	0.621+-0.01	0.759+-0.026	0.448+-0.006
Coreset	0.751+-0.016	0.733+-0.019	0.74+-0.017	0.325+-0.034	0.624+-0.012	0.608+-0.013	0.731+-0.045	0.432+-0.012
TypiClust	0.749+-0.016	0.736+-0.016	0.586+-0.038	0.348+-0.027	0.451+-0.024	0.375+-0.022	0.614+-0.046	0.397+-0.012

Table 10: AUC values for each dataset that supports query size 100.

	Splice	DNA	USPS	USPSEncoded	Cifar10Encoded	FashionMnistEnc	News
Oracle	0.803+-0.012	0.825+-0.009	0.866+-0.004	0.436+-0.057	0.749+-0.009	0.755+-0.005	0.49+-0.003
Margin	0.733+-0.024	0.711+-0.027	0.799+-0.013	0.473+-0.026	0.629+-0.012	0.628+-0.009	0.455+-0.006
Badge	0.743+-0.014	0.714+-0.032	0.804+-0.013	0.472+-0.029	0.623+-0.01	0.621+-0.01	0.456+-0.006
LeastConfident	0.715+-0.033	0.639+-0.05	0.708+-0.034	0.23+-0.034	0.631+-0.013	0.62+-0.012	0.457+-0.008
DSA	0.729+-0.021	0.697+-0.031	0.753+-0.021	0.427+-0.028	0.609+-0.013	0.546+-0.017	0.442+-0.01
BALD	0.744+-0.015	0.718+-0.024	0.765+-0.021	0.285+-0.046	0.632+-0.009	0.609+-0.01	0.444+-0.007
CoreGCN	0.742+-0.015	0.713+-0.025	0.744+-0.019	0.433+-0.032	0.583+-0.013	0.554+-0.015	0.448+-0.007
Entropy	0.733+-0.023	0.713+-0.031	0.743+-0.026	0.395+-0.037	0.618+-0.012	0.59+-0.012	0.432+-0.007
LSA	0.738+-0.017	0.716+-0.027	0.789+-0.011	0.439+-0.03	0.609+-0.013	0.608+-0.01	0.447+-0.006
Random	0.733+-0.013	0.713+-0.023	0.789+-0.012	0.468+-0.024	0.611+-0.01	0.606+-0.01	0.446+-0.005
Coreset	0.735+-0.019	0.698+-0.026	0.721+-0.021	0.396+-0.024	0.608+-0.012	0.562+-0.016	0.426+-0.012
TypiClust	0.733+-0.016	0.704+-0.025	0.592+-0.042	0.427+-0.027	0.501+-0.02	0.338+-0.02	0.383+-0.012

Table 11: AUC values for each dataset that supports query size 500.

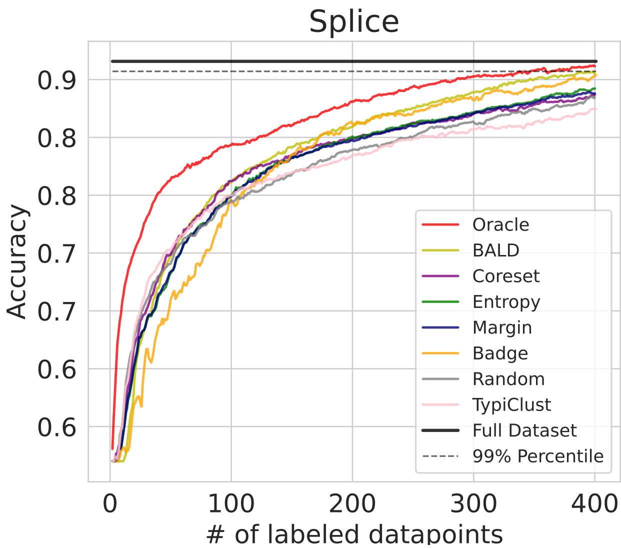
	Cifar10	FashionMnist
Oracle	0.689+-0.001	0.905+-0.001
Margin	0.556+-0.008	0.882+-0.004
Badge	0.56+-0.008	0.883+-0.005
LeastConfident	0.591+-0.01	0.884+-0.005
DSA	0.56+-0.009	0.882+-0.004
BALD	0.478+-0.014	0.878+-0.003
CoreGCN	0.553+-0.01	0.88+-0.007
Entropy	0.553+-0.009	0.882+-0.006
LSA	0.558+-0.01	0.866+-0.005
Random	0.557+-0.01	0.863+-0.005
Coreset	0.553+-0.007	0.878+-0.006
TypiClust	0.557+-0.009	0.864+-0.004

Table 12: AUC values for each dataset that supports query size 1000.

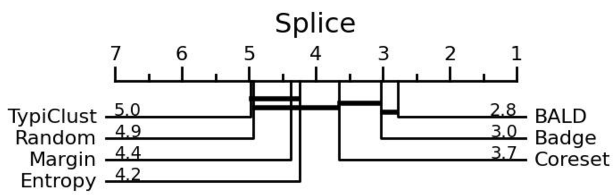
	Cifar10	FashionMnist
Oracle	0.689+-0.001	0.905+-0.001
Margin	0.56+-0.011	0.872+-0.007
Badge	0.562+-0.013	0.871+-0.007
LeastConfident	0.561+-0.012	0.873+-0.006
DSA	0.56+-0.011	0.87+-0.008
BALD	0.535+-0.011	0.866+-0.003
CoreGCN	0.557+-0.011	0.867+-0.012
Entropy	0.557+-0.014	0.871+-0.009
LSA	0.551+-0.012	0.854+-0.009
Random	0.55+-0.01	0.855+-0.006
Coreset	0.562+-0.012	0.869+-0.004
TypiClust	0.552+-0.011	0.854+-0.009

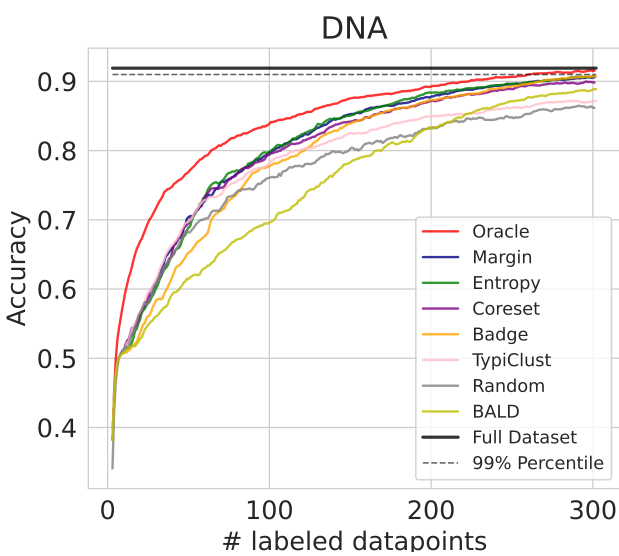
F Critical Difference Diagrams

G Individual Results

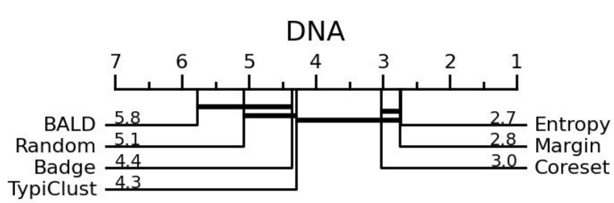


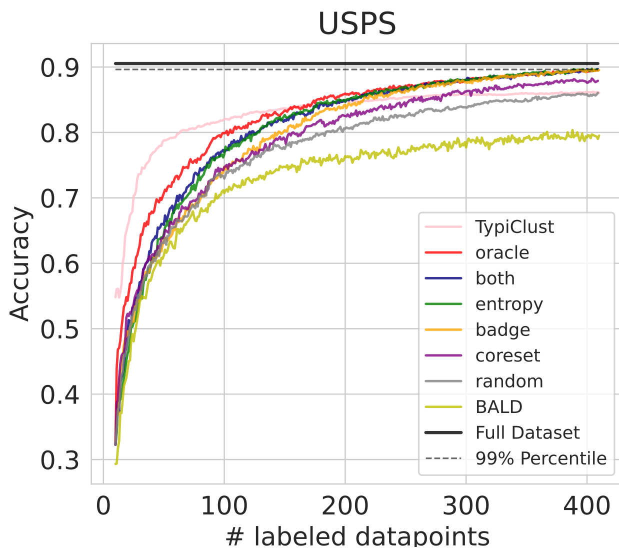
	Splice
Oracle	0.811 ± 0.010
BALD	0.785 ± 0.013
Coreset	0.778 ± 0.014
Entropy	0.774 ± 0.016
Margin	0.773 ± 0.016
Badge	0.770 ± 0.016
Random	0.768 ± 0.014
TypiClust	0.766 ± 0.014



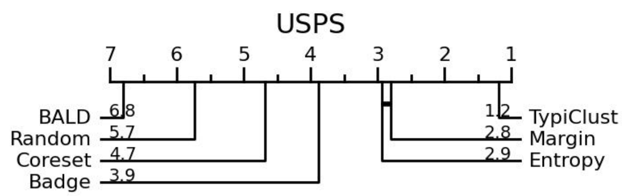


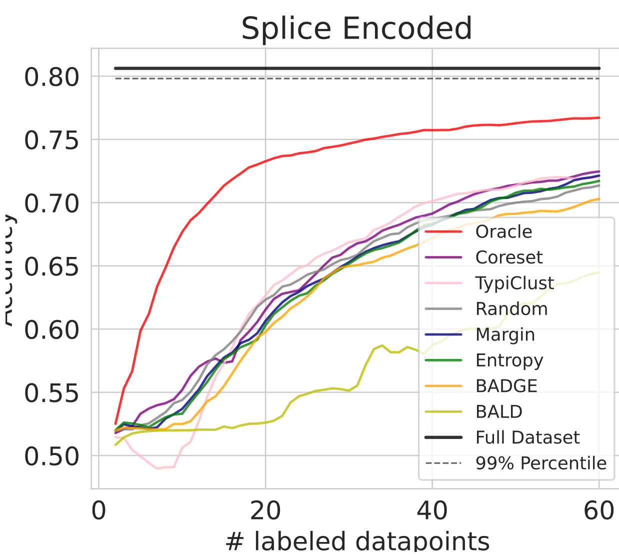
	DNA
Oracle	0.842 ± 0.021
Margin	0.807 ± 0.035
Entropy	0.805 ± 0.038
Coreset	0.796 ± 0.028
Badge	0.789 ± 0.056
TypiClust	0.788 ± 0.036
Random	0.768 ± 0.024
BALD	0.749 ± 0.044



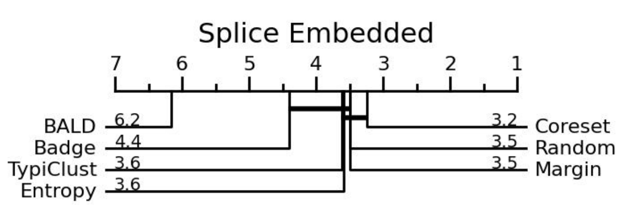


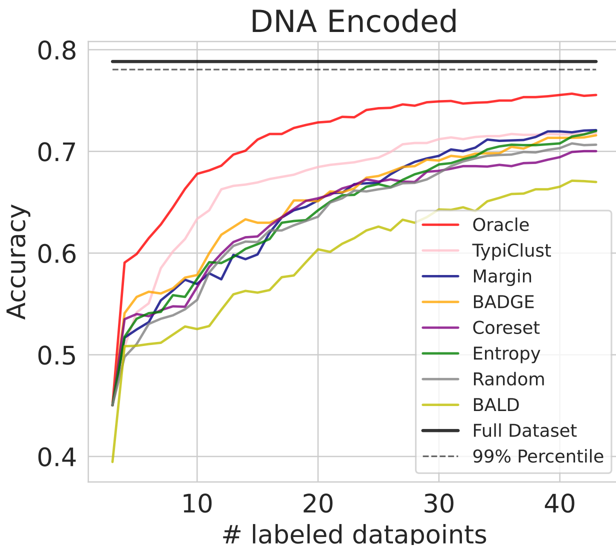
	USPS
TypiClust	0.830 ± 0.007
Oracle	0.823 ± 0.011
Margin	0.809 ± 0.013
Entropy	0.807 ± 0.013
Badge	0.795 ± 0.018
Coreset	0.788 ± 0.017
Random	0.774 ± 0.012
BALD	0.725 ± 0.050

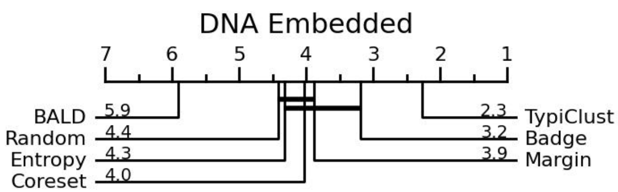




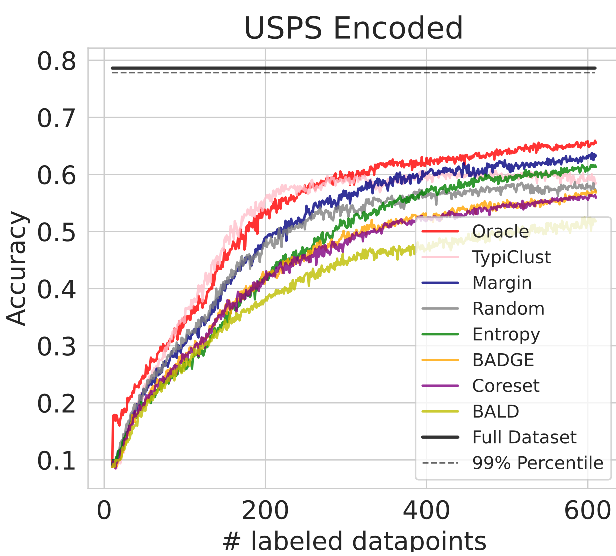
	SpliceEncoded
Oracle	0.728 ± 0.022
Coreset	0.648 ± 0.027
TypiClust	0.645 ± 0.042
Random	0.643 ± 0.036
Entropy	0.636 ± 0.033
Margin	0.636 ± 0.033
Badge	0.627 ± 0.040
BALD	0.565 ± 0.049

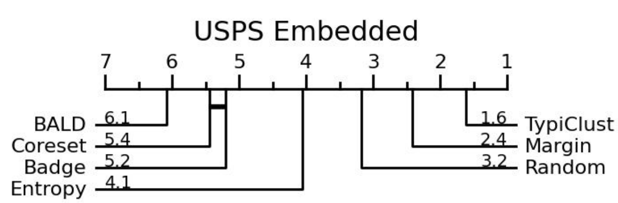




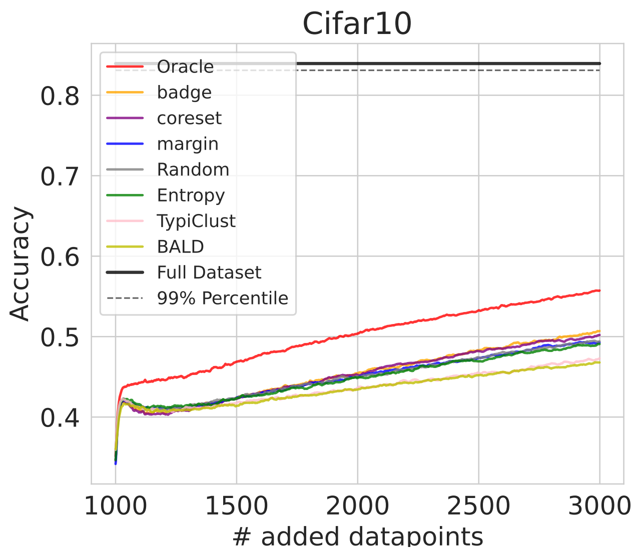


	DNAEncoded
Oracle	0.709 ± 0.023
TypiClust	0.672 ± 0.029
Margin	0.648 ± 0.047
Badge	0.647 ± 0.037
Coreset	0.640 ± 0.041
Entropy	0.629 ± 0.062
Random	0.626 ± 0.035
BALD	0.594 ± 0.039

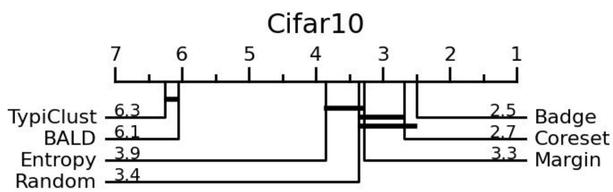


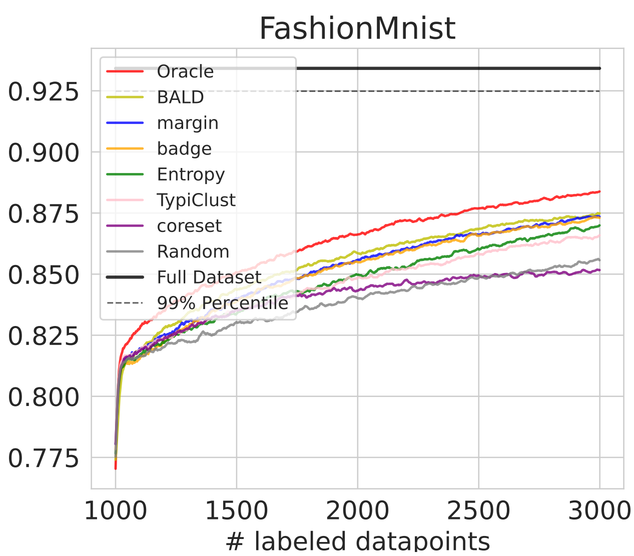


	USPSEncoded
Oracle	0.522 ± 0.021
TypiClust	0.507 ± 0.025
Margin	0.496 ± 0.030
Random	0.468 ± 0.025
Entropy	0.459 ± 0.021
Badge	0.440 ± 0.026
Coreset	0.435 ± 0.027
BALD	0.402 ± 0.052

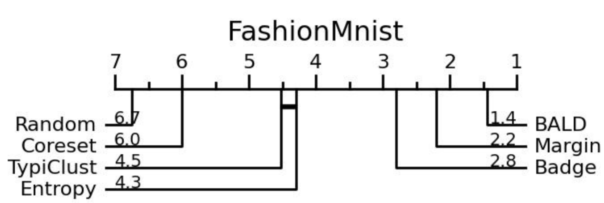


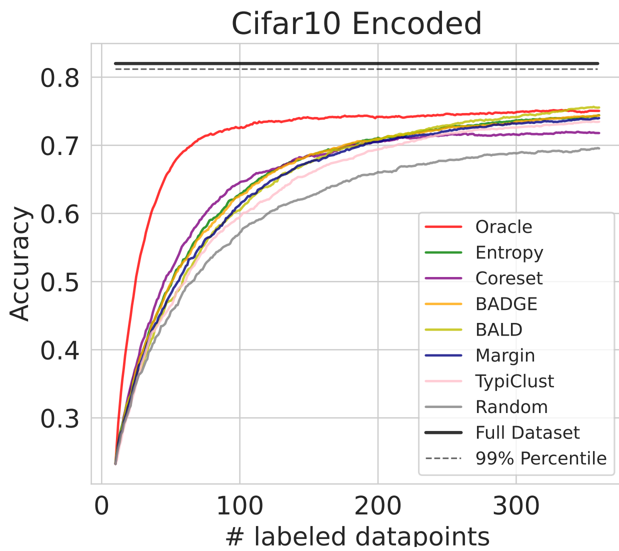
	Cifar10
Oracle	0.500 ± 0.010
Badge	0.453 ± 0.012
Coreset	0.453 ± 0.009
Margin	0.451 ± 0.010
Random	0.450 ± 0.012
Entropy	0.449 ± 0.010
TypiClust	0.436 ± 0.010
BALD	0.436 ± 0.010



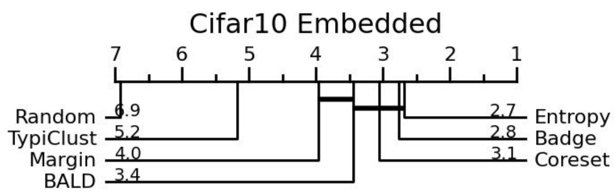


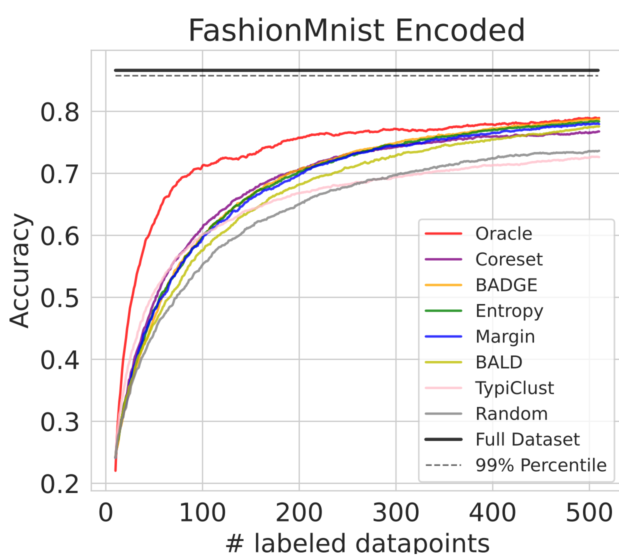
	FashionMnist
Oracle	0.862 ± 0.003
BALD	0.854 ± 0.003
Margin	0.851 ± 0.003
Badge	0.851 ± 0.003
Entropy	0.847 ± 0.004
TypiClust	0.846 ± 0.004
Coreset	0.840 ± 0.004
Random	0.837 ± 0.004



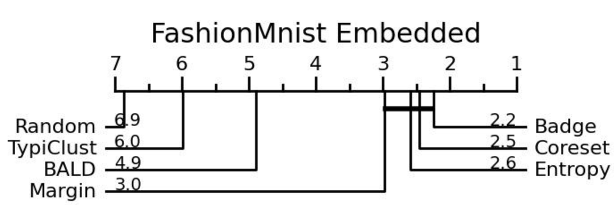


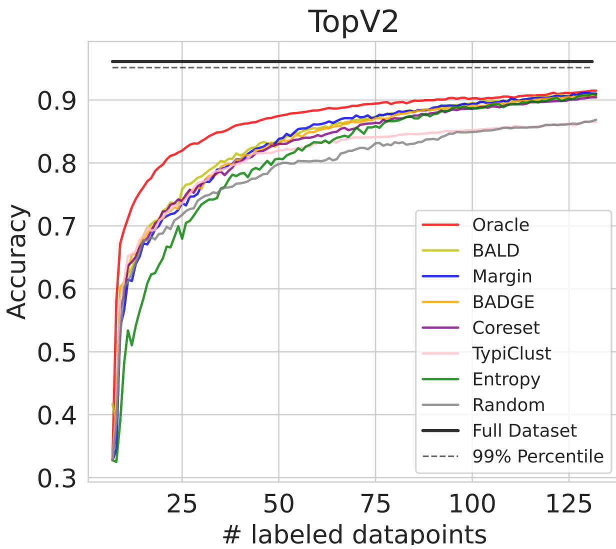
	Cifar10Encoded
Oracle	0.714 ± 0.007
Entropy	0.654 ± 0.013
Coreset	0.653 ± 0.012
Badge	0.653 ± 0.012
BALD	0.650 ± 0.016
Margin	0.647 ± 0.012
TypiClust	0.636 ± 0.009
Random	0.607 ± 0.013



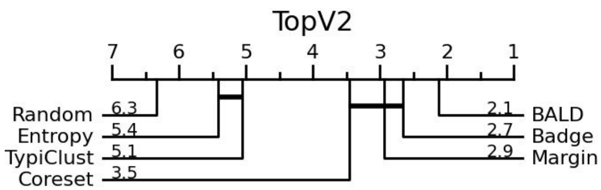


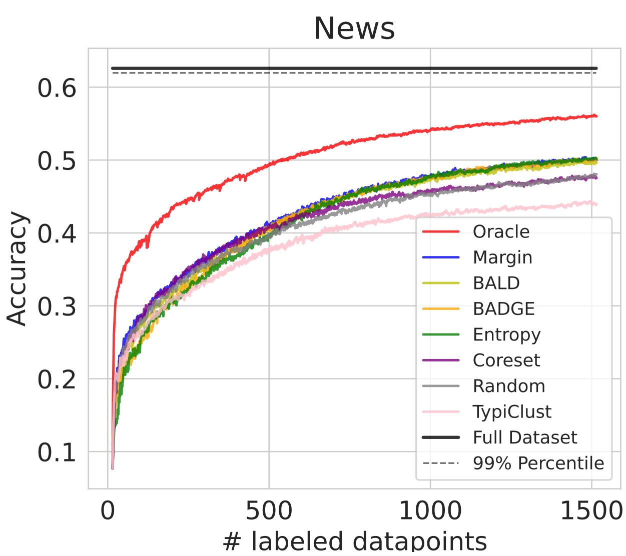
	FashionMnistEncoded
Oracle	0.732 ± 0.006
Coreset	0.686 ± 0.008
Badge	0.685 ± 0.008
Entropy	0.684 ± 0.009
Margin	0.682 ± 0.011
BALD	0.668 ± 0.009
TypiClust	0.652 ± 0.009
Random	0.640 ± 0.011



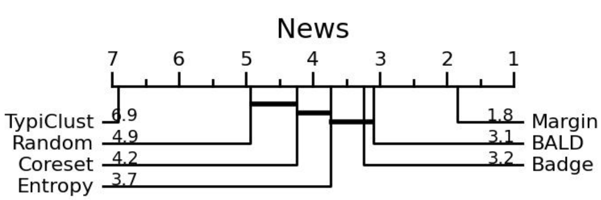


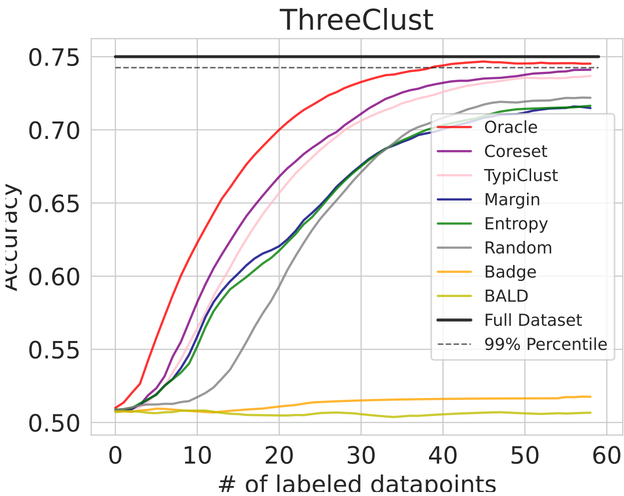
	TopV2
Oracle	0.862 ± 0.006
BALD	0.831 ± 0.013
Badge	0.826 ± 0.015
Coreset	0.823 ± 0.016
Margin	0.822 ± 0.015
TypiClust	0.805 ± 0.015
Entropy	0.801 ± 0.025
Random	0.787 ± 0.015



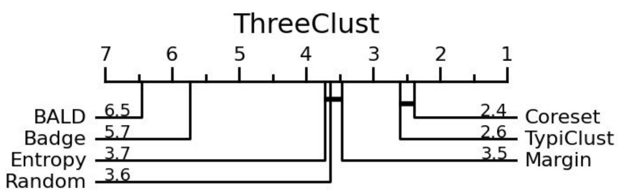


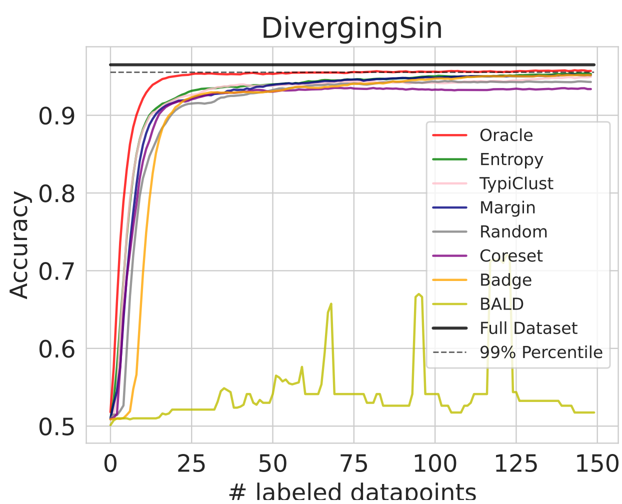
	News
Oracle	0.502 ± 0.005
Margin	0.427 ± 0.007
BALD	0.421 ± 0.008
Badge	0.420 ± 0.011
Entropy	0.416 ± 0.010
Coreset	0.415 ± 0.011
Random	0.409 ± 0.008
TypiClust	0.385 ± 0.010



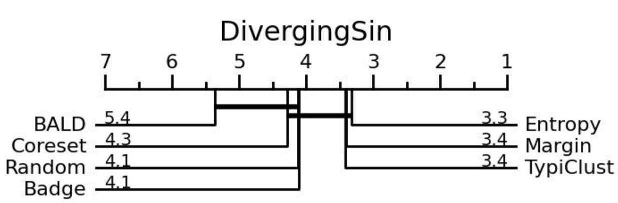


	ThreeClust
Oracle	0.722 ± 0.097
Coreset	0.698 ± 0.058
TypiClust	0.697 ± 0.055
Entropy	0.682 ± 0.098
Random	0.672 ± 0.067
Margin	0.669 ± 0.095
Badge	0.524 ± 0.086
BALD	0.507 ± 0.050





	DivergingSin
Oracle	0.948 ± 0.198
Entropy	0.936 ± 0.202
TypiClust	0.930 ± 0.196
Margin	0.929 ± 0.201
Random	0.919 ± 0.191
Badge	0.914 ± 0.202
Coreset	0.914 ± 0.197
BALD	0.661 ± 0.167



H Seeding Strategy

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, only [10] has described and implemented a fully reproducible pipeline for AL evaluation. The term reproducibility in this work is used as a synonym not only for the reproducibility of an experiment (a final result given a seed), but also the reproducibility of all subsystems independent of each other. The seed for one subsystem should always reproduce the behavior of this subsystem independent of all other subsystems and their seeds. The main obstacle for ensuring reproducibility is the seeding utility in PyTorch, Tensorflow, and other frameworks, whose default choice is a single global seed. Since many subsystems 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, data split or the order of training batches. Even though some workarounds exist, e.g. 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 separate Random Number Generators (RNGs) 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 ongoing 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, 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 (e.g. 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 J.

Note: Even though we decoupled the subsystems via the described seeds, the subsystems can still influence each other in a practical sense. For example, keeping $s_\mathcal{D}$ fixed does not mean that always the same sequence of samples from $\mathcal{U}$ (if subsamples are drawn) are shown to all acquisition functions. This is practically impossible, as different acquisition functions pick different $x^{(i)}$. However, the hypothetical **tree** of all possible sequences of samples from $\mathcal{U}$ remains the same, granting every acquisition function equal possibilities.

I Oracle Curve Forecasting

TODO

J Hyperparameters per AL Algorithm

Table 13: Selected hyperparameters for all tested acquisition functions. Last column indicates the source of our implementation.

Algorithm	Sample Size	Other	Source
BADGE	100		Based on [1, 14]
BALD	100	Dropout Trials: 5	Based on [4]
Coreset	8000		Own
TypiClust	10000	Min Cluster Size: 5 Max # Clusters: 500	Based on [8]
Margin	8000		Own
Entropy	8000		Own

K Hyperparameters and Preprocessing per Dataset

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 in the experiment run itself and depends on the dataset-seed.

Tabular: We use **Splice**, **DNA** and **USPS** from LibSVMTools [20]. All three datasets are normalized between [0, 1].

Image: We use **FashionMNIST** [25] and **Cifar10** [13], since both are widely used in AL literature. Both datasets are normalized according to their standard protocols.

Text: We use **News Category** [18] and **TopV2** [6]. 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 [21]. Since neither dataset provided a fixed test set, we randomly split 7000 datapoints into a test set.

Dataset	Seed Set	Budget	Val Split
Splice	1	400	0.2
SpliceEnc.	1	60	0.2
DNA	1	300	0.2
DNAEnc	1	40	0.2
USPS	1	400	0.2
USPSEnc	1	600	0.2
FashionMnist	100	2000	0.04
FashionMnistEnc	1	500	0.04
Cifar10	100	2000	0.04
Cifar10Enc	1	350	0.04
TopV2	1	125	0.25
News	1	1500	0.03

Table 14: Size of the seed set is given by number of labeled sample per class.

Dataset	Classifier	Optimizer	LR	Weight Decay	Dropout	Batch Size
Splice	[24, 12]	NAdam	1.2e-3	5.9e-5	0	43
SpliceEnc.	linear	NAdam	6.2e-4	5.9e-6	0	64
DNA	[24, 12]	NAdam	3.9e-2	3.6e-5	0	64
DNAEnc	linear	NAdam	1.6e-3	4e-4	0	64
USPS	[24, 12]	Adam	8.1e-3	1.5e-6	0	43
USPSEnc	linear	NAdam	7.8e-3	1.9e-6	0	64
FashionMnist	ResNet18	NAdam	1e-3	0	0	64
FashionMnistEnc	linear	Adam	1.6e-3	1e-5	5e-2	64
Cifar10	ResNet18	NAdam	1e-3	0	0	64
Cifar10Enc	linear	NAdam	1.7e-3	2.3e-5	0	64
TopV2	BiLSTM	NAdam	1.5e-3	1.7e-7	5e-2	64
News	BiLSTM	NAdam	1.5e-3	1.7e-7	5e-2	64

Table 15: Classifier architectures and optimized hyperparameters per dataset. Numbers in brackets signify a MLP with corresponding hidden layers.

L AL Pseudocode

Algorithm 1 Active Learning Loop

Require: $\mathcal{L}, \mathcal{U}, \mathcal{D}_{\text{test}}, \text{Train}, \text{Seed}, \hat{y}$
Require: Ω ▷ Acquisition Function
1: $\mathcal{L}^{(1)} \leftarrow \text{Seed}(\mathcal{U})$ ▷ Create the initial labeled set
2: $\mathcal{U}^{(1)} \leftarrow \mathcal{U}$
3: **for** $i := 1 \dots B$ **do**
4: $\quad \text{acc}^{(i)} \leftarrow \text{Train}(\mathcal{L}^{(i)})$
5: $\quad a^{(i)} \leftarrow \Omega(\mathcal{U}^{(i)})$
6: $\quad \mathcal{L}^{(i+1)} \leftarrow \mathcal{L}^{(i)} \cup \{(\mathcal{U}_a^{(i)}, A(\mathcal{U}_a^{(i)}))\}$
7: $\quad \mathcal{U}^{(i+1)} \leftarrow \mathcal{U}^{(i)} \setminus \{\mathcal{U}_a^{(i)}\}$
8: **return** $\frac{1}{B} \sum_{i=1}^{B} \text{acc}^{(i)}$

Algorithm 2 Retrain

Require: $\mathcal{L}, \mathcal{D}_{\text{val}}, \mathcal{D}_{\text{test}}$
Require: $\hat{y}, e_{\text{max}}$
1: $\text{loss}^* \leftarrow \infty$
2: **for** $i := 1 \dots e^{\text{max}}$ **do**
3: $\quad \hat{y}_{i+1} \leftarrow \hat{y}_i - \eta \nabla_{\hat{y}} \ell(\mathcal{L}, \hat{y})$
4: $\quad \text{loss}_i \leftarrow \ell(\mathcal{D}^{\text{val}}, \hat{y})$
5: $\quad$ **if** $\text{loss}_i < \text{loss}^*$ **then**
6: $\qquad \text{loss}^* \leftarrow \text{loss}_i$
7: $\quad$ **else**
8: $\qquad$ Break
9: **return** $\text{Acc}(\mathcal{D}^{\text{test}}, \hat{y})$

Algorithm 3 Acquire Oracle Ω

Require: $\mathcal{U}, \mathcal{L}, A, \mathcal{D}_{\text{test}}, \tau, \hat{y}_\theta$
Require: Train, Margin, Acc
1: $\text{acc}^0 \leftarrow \text{acc}^* \leftarrow \text{Acc}(\mathcal{D}_{\text{test}}, \hat{y}_\theta)$
2: **for** $k := 1 \dots \tau$ **do**
3: $\quad u_k = \text{unif}(\mathcal{U})$
4: $\quad \mathcal{L}' \leftarrow \mathcal{L}^{(i)} \cup \{(u_k, A(u_k))\}$
5: $\quad \hat{y}'_\theta \leftarrow \text{Train}(\mathcal{L}', \hat{y}_\theta)$
6: $\quad \text{acc}' \leftarrow \text{Acc}(\mathcal{D}_{\text{test}}, \hat{y}'_\theta)$
7: $\quad$ **if** $\text{acc}' > \text{acc}^*$ **then**
8: $\qquad \text{acc}^* \leftarrow \text{acc}'$
9: $\qquad u^* \leftarrow u_k$
10: **if** $\text{acc}^0 = \text{acc}^*$ **then**
11: $\quad u^* \leftarrow \text{Margin}(\mathcal{U}, \hat{y}_\theta)$
return u^*

Alg. 3 replaces the acquisition function Ω in the AL loop (Alg. L line 5).