University of Groningen

Active learning for reducing labeling effort in text classification tasks

Jacobs, Pieter Floris; Wenniger, Gideon Maillette de Buy; Wiering, Marco; Schomaker, Lambert

Active learning for reducing labeling effort in text classification tasks

Pieter Floris Jacobs[1],
Gideon Maillette de Buy Wenniger[2,1][0000−0001−8427−7055],
Marco Wiering[1][0000−0003−4331−7537],
Lambert Schomaker[1][0000−0003−2351−930X]

[1] University of Groningen, Groningen, The Netherlands
`p.f.jacobs AT student.rug.nl;`
`{l.r.b.schomaker, m.a.wiering } AT rug.nl`
[2] Open University of the Netherlands
`gemdbw AT gmail.com`

Abstract. Labeling data can be an expensive task as it is usually performed manually by domain experts. This is cumbersome for deep learning, as it is dependent on large labeled datasets. Active learning (AL) is a paradigm that aims to reduce labeling effort by only using the data which the used model deems most informative. Little research has been done on AL in a text classification setting and next to none has involved the more recent, state-of-the-art NLP models. Here, we present an empirical study that compares different uncertainty-based algorithms with $BERT_{base}$ as the used classifier. We evaluate the algorithms on two NLP classification datasets: Stanford Sentiment Treebank and KvK-Frontpages. Additionally, we explore heuristics that aim to solve presupposed problems of uncertainty-based AL; namely, that it is unscalable and that it is prone to selecting outliers. Furthermore, we explore the influence of the query-pool size on the performance of AL. Whereas it was found that the proposed heuristics for AL did not improve performance of AL; our results show that using uncertainty-based AL with $BERT_{base}$ outperforms random sampling of data. This difference in performance can decrease as the query-pool size gets larger.

Keywords: Active Learning · Text Classification · Deep Learning · BERT.

1 Introduction

Deep Learning (DL) is a field in machine learning in which neural networks with a large number of layers are made to perform complicated human tasks. These networks have to be trained on a large amount of data to be able to learn the underlying distribution of the task they are trying to model. In supervised learning, this data is required to be labeled with the desired output. This allows the network to learn to map the input to the desired output. This study will focus on an instance of supervised learning, called text classification. Data labeling is

usually done manually and can grow to be an expensive and time-consuming task for larger datasets, like those used in DL. This begs the question of whether there is no way to reduce the labeling effort while preserving good performance on the chosen task. Similarly to lossy compression [1], we want to retain a good approximation of the original dataset while at the same time reducing its size as much as possible. More specifically: given a training set, how can we optimally choose a limited number of examples based on the amount of relevant information they contain for the target task?

Conceptually, answering this question requires quantifying the amount of information contained in each datapoint. This finds its roots, like lossy compression, in information theory [26]. A model trained on limited data has an entropy associated with its target variable predictions. Our goal is to greedily select the data for labeling, while reducing entropy as much as possible, similar to how it is done in research on decision trees [11]. In essence, we aim to incrementally, optimally select a subset of datapoints; such that the distribution encoded by the learned model maximizes the information gain or equivalently minimizes the Kullback-Leibler divergence [18] with respect to the unknown distribution of the full labeled data. However, there are two problems. First, the labels of the data are not known until labeling, and additional heldout labeled data to aid the selection is typically not available either. This contrasts with the easier case of summarizing a known dataset by a subset of data, in which the Kullback-Leibler divergence of a selected subset with the full set can be measured and minimized. Second, because the parameters of a neural network change during training, predictions and certainty of new datapoints also change. Because of these two problems, examples can only be greedily selected based on their expected utility for improving the current, incrementally improved model. As the actual labels for examples are lacking before their selection, their real utility cannot be known during selection. Therefore, only proxies for this utility such as model uncertainty can be used, as discussed next.

A machine-learning technique called Active Learning (AL) [25] can be used to combat these problems. In AL, a human labeler is queried for datapoints that the network finds most informative given its current parameter configuration. The human labeler assigns labels to these queried datapoints and then the network is retrained on them. This process is repeated until the model shows robust performance, which indicates that the data that was labeled is a sufficient approximation of the complete dataset. There are multiple types of informativeness by which to determine what data to query the oracle for. For instance calculating what results in the largest model change [3] or through treating the model as a multi-arm bandit [2]. However, the existing literature predominantly utilizes different measures of model uncertainty [5,7,8,9,31], which is also done in this research. Bayesian probability theory provides us with the necessary mathematical tools to reason about uncertainty, but for DL has its complications. The reason is that (typical) neural networks, as used for classification and regression, are discriminative models. These produce a single output, a so called point estimate. Even in the case of softmax outputs this is not a true probability den-

sity function [7,8]. Another view on this is that modern neural networks often lack adequate *confidence calibration*, meaning they fail at predicting probability estimates representative of the true correctness likelihood [12].

This poses a problem to Bayesian probability theory as it prevents us from being able perform Bayesian inference. With Bayesian inference we can determine the probability of a certain output y* given a certain input point x*:

$$p(y*|x*, X, Y) = \int p(y*|x*, \omega)p(\omega, X, Y)d\omega \tag{1}$$

Unfortunately, for the discriminative neural network models there is no probability distribution: the output is always the same for a given input. What is more, even if we suppose the network was generative (Eq. 1), the integral is not analytically solvable due to the fact that we need to integrate over all possible parameter settings ω. However, it can be approximated. Existing literature has explored different methods of achieving this, with Monte Carlo Dropout (MCD) being the most popular one [5,8,32]. In MCD, the network applies dropout [29] to make the network generative. Multiple stochastic forward passes are performed to produce multiple outputs for the same input. The outputs can then be used to summarize the uncertainty of the model in a variety of ways.

This research uses the MCD approximation to compare different uncertainty-related AL query methods for text classification, noting there is still little literature on the usability of AL for modern NLP models. We strive to answer the following research question:

Research Question. *How can uncertainty-based Active Learning be used to reduce labeling effort for text classification tasks?*

Where previous literature focused on comparing AL strategies on small datasets and on the test accuracy of the final classifier, this paper will try and explore the usability of AL on a real-world setting, in which factors like the effect of transfer learning and considerations such as scalability have to be taken into account. The goal is to reach a performance similar to the state-of-the-art text-classification models that use a large randomly sampled set of labeled examples as training set. This should show whether AL can be applied to reduce labeling effort.

2 Methods

This section will go on to describe the general AL loop, the model architecture, the used query functions, the implemented heuristics, and finally the experimental setup.

2.1 Active Learning

An implementation of the general AL loop/round is shown in Appendix A.3 (Algorithm 1). It consists of four steps:

1. **Train:** The model is reset to its initial parameters. After this, the model is trained on the labeled dataset $\mathcal{L}$. The model is reset before training because otherwise the model would overfit on data from previous rounds [15].
2. **Query:** A predefined query function is used to determine what data is to be labeled in this AL round. As discussed, this can be done in various ways, but the guiding principle is that the data that the model finds most useful for the chosen task gets queried.
3. **Annotate:** The queried data is parsed to a human expert, often referred to as the oracle. The oracle then labels the queried examples.
4. **Append:** The newly-labeled examples are transferred from the unlabeled dataset $\mathcal{U}$ to $\mathcal{L}$. The model is now ready to be retrained to recompute the informativeness of the examples in $\mathcal{U}$ now that the underlying distribution of $\mathcal{L}$ has been altered.

Please note that the datasets used for the experiments (Section 2.5) were fully labeled and the annotation step thus got skipped in this research. $\mathcal{U}$ existed out of labeled data that was only trained on from the moment it got queried. This was done to speed up the process and to enable scalable and replicable experiments with varying experimental setups.

2.2   Model Architecture

BERT The model used to classify the texts was BERT_{base} [4], a state-of-the-art language model which is a variant of the Transformer model [33]. Specifically, we used the uncased version of BERT_{base}, as the information of capitalization and accent markers was judged to be not helpful for the used tasks and datasets. Due to computational constraints, only the first sentence of the used texts was put into the tokenizer and the maximal length to which the tokenizer either padded or cut down this sentence was set to 50. To better deal with unknown words and shorter text, we used the option of the BERT_{base} tokenizer to make use of special tokens for sentence separation, padding, masking and to generalize unknown vocabulary. Finally, a softmax layer was added to the end of BERT_{base}, which is essential as the implemented query functions (Section 2.3) compute uncertainty based on sampled output probability distributions.

Monte Carlo Dropout Monte Carlo dropout (MCDO) is, as discussed in Section 1, a technique that enables reasoning about uncertainty with neural networks. Dropout [29] essentially 'turns off' neurons during the forward pass with a predefined probability. Dropout is normally used during training to prevent overfitting and create a more generalized model. In MCDO though, it is used to approximate Bayesian inference [8] through creating T predictions for all datapoints, using T slightly different models induced by different dropout samples. The result of these so-called stochastic forward passes (SFP's) can then be used by the query function to compute the uncertainty, as will be explained in Section 2.3. The way MCDO is incorporated in the AL loop is shown in green in the Appendix (Algorithm 2). BERT_{base} has two different types of dropout layers:

hidden dropout and attention dropout. Both were turned on when performing a stochastic forward pass. Note that there are other ways of approximating Bayesian inference with neural networks. Frequently used ones are:

– Having an ensemble of neural networks vote on the label [17].
– Monte Carlo Batch Normalization (MCBN) [31].

MCDO was chosen over the ensemble method due to it being easier to implement and quicker to train. MCBN was not chosen as it has been shown to be more inconsistent than MCDO [5].

SentenceBERT Textual data offers the advantage of having access to the use of pre-trained word embeddings. These are learned representations of words into a vector space in which semantically similar words are close together. Textual embeddings can be computed in a variety of ways. BERT specific ones include averaging the pooled BERT embeddings and looking at the BERT CLS token output. Other more general ways are averaging over Glove word embeddings [22] and averaging embeddings created by a Word2Vec model [20]. We have opted to make use of SentenceBERT [23], a Siamese BERT architecture trained to produce embeddings that can be adequately compared using cosine-similarity. For our purposes this provides better performance than the other embedding computations. SentenceBERT was used separately from the previously discussed $BERT_{base}$ model, and was used only for assigning embeddings to each sentence in the dataset that were used by the heuristics described in Section 2.4.

2.3   Query Functions

The query functions determine the model its data selection choices in the AL loop. This paper will focus on functions that reason about uncertainty, obtained from approximated Bayesian distributions [8]. For every datapoint, the distribution is derived from T stochastic forward passes and resulting T (in our case) softmax probability distributions. The three implemented query functions are:

1. Variation ratio
2. Predictive entropy
3. Bayesian active learning by disagreement (BALD).

These query functions are discussed in more detail Appendix A.1 as well as by Gal [7].

2.4   Heuristics

Redundancy Elimination In AL, a larger query-pool size (from now on referred to as q) results in the model being retrained less and the uncertainties of examples being re-evaluated less frequently. Consequently, the model gets to make less informed decisions as it uses less up-to-date uncertainty estimates. Larger q could therefore theoretically cause the model to collect many similar

examples for specific example types with high model uncertainty in an AL round. Say for instance we were dealing with texts about different movie genres. Suppose the data contained a lot of texts about the exact same movie. When the model would be uncertain about this type of text, a large q would result in a large amount of these texts getting queried. This could be wasteful, as querying this type of text a small amount of times would likely result in the model no longer being uncertain about that type of text. Note however, that low model uncertainty by itself is no guarantee for robustly making accurate predictions for a type of examples. Yet provided such robust performance is achieved, additional examples of the same type would be a waste.

The above could form a problem as although a smaller q should theoretically provide us with better results, it also requires more frequent uncertainties re-computation. Every computation of the uncertainties requires T stochastic forward passes on the unlabeled dataset $\mathcal{U}$. This entails that, next to the computation, the time required to label a dataset would increase as well, which is not in line with our goal. In hopes of improving performance with larger q, we propose two heuristics:

1. Redundancy Elimination by Training (RET)
2. Redundancy Elimination by Cosine Similarity (RECS)

For both of these heuristics, a new pool, which we will refer to as the redundancy-pool $\mathcal{RP}$, is introduced. The query-pool $\mathcal{QP}$ will be a subset of $\mathcal{RP}$ of which we will try to select the most dissimilar examples.

RET tries to eliminate redundant data out of $\mathcal{RP}$ by using it as a pool to retrain on. The datapoint with the highest uncertainty is trained on for one epoch and then the uncertainties of the examples in $\mathcal{RP}$ are recomputed. This process gets repeated until $\mathcal{QP}$ is of the desired size. Note that although this strategy seems similar to having a q of one, it is less computationally expensive as only the uncertainties for the examples in $\mathcal{RP}$ have to be recomputed (which also shrinks after each repetition). Algorithm 3 of Appendix A.3 shows how RET is integrated in the AL loop.

The main purpose of RET is to enable the use of larger q. However, one needs to be mindful of the fact that when q is increased, $\mathcal{RP}$ is to be increased in size well. This being due to the fact that smaller differences between the sizes of $\mathcal{RP}$ and $\mathcal{QP}$ result in less influence of the heuristic. In the RET algorithm, forward passes over $\mathcal{RP}$ contribute to the total amount of forward passes. Furthermore, this contribution increases linearly with the redundancy-pool size ($|\mathcal{RP}|$) and in practice coupled query-pool size q. Using $|\mathcal{RP}| = 1.5 \times q$, this contribution starts to dominate the total amount of forward passes (approximately) once $q > \sqrt{|\text{data}|}$. This is explained in more detail in Appendix A.2. This limits its use for decreasing computation by increasing q. Because of this, RECS is aimed at being computationally cheaper.

Instead of retraining the model and constantly taking into account recomputed uncertainties, RECS makes use of the sentence embeddings created by SentenceBERT (Section 2.2). The assumption made is that semantically similar

data conveys the same type of information to the model. The examples are selected based on their cosine similarity to other examples. $\mathcal{RP}$ is looped through and examples are only added to $\mathcal{QP}$ if their cosine similarity to all other points that are already in $\mathcal{QP}$ is lower than the chosen threshold l. If not enough examples are selected to get the desired q, the threshold gets decreased by 0.01. Algorithm 4 of Appendix A.3 shows how this heuristic is added to the AL loop.

Sampling by Uncertainty and Density (SUD) Schomaker and Oosten [21] showed that the distinction between separability and prototypicality is important to account for. In their use case of the SVM, datapoints that had a high margin to the decision boundary were not always representative of the class prototype. Uncertainty sampling also tries to sample examples close to the decision boundary, but has been shown to often select outliers [24,30]. Outliers contain a lot of information that the model has not encountered yet, but this information is not necessarily useful. As with the previously described RECS heuristic, we hypothesize that semantically similar sentences provide the same type of information. In that situation, outliers are very far from other examples in embedding space.

Zhu et. al [34] proposed a K-Nearest-Neighbor-based density approach called Sampling by Uncertainty and Density (SUD) to avoid outliers based on their distance in embedding space. In this approach, the mean cosine similarity between every datapoint and its K most similar neighbors is computed. A low value indicates that a datapoint is not very similar to others. This value is then multiplied with the uncertainty and the dataset is sorted based on this Uncertainty-Density measure. They showed that this measure improved performance of the maximum entropy model classifier. We will explore whether this approach also works for BERT combined with the embeddings computed by SentenceBERT. The adjusted pseudocode is shown in Appendix A.3 (Algorithm 5).

2.5   Experimental Setup[3]

Data Two datasets were used to validate and compare the performance of the different AL implementations. Table 1 shows an overview of the amount of examples and classes of each dataset. The first of the used datasets was the Stanford

Table 1. An overview of the two datasets used in the experiments

Dataset	Examples	Number of Classes
SST	11,850	5
KvK	2212	15

---

[3]The code used for the experiments can be found at https://github.com/Pieter-Jacobs/bachelor-thesis

Sentiment Treebank [28] (SST). SST exists out of 215,154 phrases from movies with fine-grained sentiment labels in the range of 0 to 1. These phrases are contained in the parse trees of 11,855 sentences. Only these full sentences were used in the experiments, and the sentiment labels were mapped to five categories in the following way:

- $0 \leq$ label < 0.2: very negative
- $0.2 \leq$ label < 0.4: negative
- $0.4 \leq$ label ≤ 0.6: neutral
- $0.6 <$ label ≤ 0.8: positive
- $0.8 <$ label ≤ 1: very positive

The second dataset that was used consists of the of descriptions of companies located in Utrecht. The companies are all registered at the Dutch Chamber of Commerce, or Kamer van Koophandel (KvK) and were mapped to their corresponding SBI-code. The SBI code denotes the sector a company operates in, as defined by the KvK. The HTML of the companies websites was scraped and the meta content that was tagged as the description was extracted. In nearly all cases, this contained a short description about what the company was involved in. This dataset will not be shared and is not available online due to the fact that it was constructed as part of an internship at Dialogic.

Evaluation Metrics To evaluate and compare the performance of the different AL strategies, two evaluation metrics were reported: the accuracy and an altered version of the deficiency metric proposed in [34].

The variant of deficiency that was used is shown in Eq. 2, in which n denotes the amount of accuracy scores, $acc(R)$ denotes the accuracy of the reference strategy and $acc(C)$ the accuracy of the strategy to be compared to this reference strategy. Instead of using the accuracy that was achieved in the final AL round like in [34], we use the overall maximum accuracy. This accounts for the fact that the last achieved accuracy in a classification task is not necessarily the best value, while still returning a metric which provides a summary of the entire learning curve. A value of <1 indicates a better performance than the reference strategy whereas a value of >1 indicates a worse performance.

$$DEF(AL, R) = \frac{\sum_{t=1}^{n}(max(acc(R)) - acc_t(C)}{\sum_{t=1}^{n}(max(acc(R)) - acc_t(R))} \qquad (2)$$

Experiments The goal of the experiments was to answer the question of whether overall labeling effort could be reduced through making use of AL. We split this into the following three sub-questions:

1. Does AL achieve better performance with less data when compared to plain random sampling?
2. What is the relation between query-pool size q and the achieved performance?
3. Do the proposed heuristics (SUD, RET, RECS) improve the performance of AL?

Table 2. The statistical setup used for both datasets. The percentages used are relative to the full dataset size.

Dataset	Seed	$\mathcal{U}$	Dev	Test
SST	594 (5%)	7951 (67%)	1101 (9%)	2210 (19%)
KvK	111 (5%)	1659 (75%)	221 (10%)	1659 (10%)

The statistical setup used for the experiments can be found in Table 2. The setup for SST was based on the proposed setup in [28]. To reiterate, the following AL strategies were implemented:

1. Variation Ratio
2. Predictive Entropy
3. BALD

4. RET (Section 2.4)
5. RECS (Section 2.4)
6. SUD (Section 2.4)

To answer subquestion 1, these strategies were compared to the performance of random sampling using a q of 1% of the dataset size. For subquestion 2, the three query functions were be compared across three q: 0.5%, 1% and 5% of the dataset size. Finally, to be able to answer subquestion 3, RET, RECT and SUD were compared with a q of 1%. As RET, RECS and SUD were meant as additions to general problems of uncertainty-based AL, they were only tested for the variation ratio query function. This function was chosen, because it was reported in [7] to give the best result. To make the results more generalizable, all the experiments mentioned above were run three times.

Moreover, to test the assumption of the RECT strategy, we measured whether there was a relation between how the model softmax predictions changed towards the one-hot vector of the actual label and the cosine similarity to the datapoint that was trained on. The relationship was quantified by means of Kendall's τ between the ranking of the examples based on which one had the largest change in KL divergence after training on the top example and the ranking of the examples based on cosine similarity to the example being trained on.

Hyperparameters Table 4 gives an overview of used hyperparameters. Both dropout rate and l (the cosine-similarity threshold used in RECS) were chosen based on a grid search across both datasets. The amount of stochastic forward passes T was based on [6] and was set to 10 across all experiments.[4] Early stopping was applied on each training phase of the AL loop, Table 3 shows the amount of epochs used for each dataset. The model yielding the lowest validation loss across all epochs was used for evaluation and uncertainties computation. Note that in a normal AL setting, validation sets are usually not available due to the labelling effort required and this strategy would be less feasible.

The Adam algorithm [16] was used for optimization and its learning rate was tuned based on the CLR method [27]. The best performing computationally

---

[4]Larger values up to 100 were tested, but induced much larger training times without noteworthy performance gains.

Table 4. Hyperparameters values

Parameter	Value
Dropout rate	0.2
T	10
l	0
β_1, β_2	0.9, 0.999
ϵ	$1 * 10^{-8}$
Learning rate	$2 * 10^{-5}$
Batch size	128
$\mathcal{RP}$ size	$1.5*q$
Embedding dim.	768

Table 3. The amount of epochs used for early stopping for the different datasets.

Dataset	# Epochs
SST	15
KvK	25

feasible batch size (128), out of the tried batch sizes (32, 64, 128, 256), was used in all experiments. The betas and ϵ were set to their default values. The size of $\mathcal{RP}$ was chosen arbitrarily, determining its optimal choice is left future research.

Finally, dimensionality reduction using PCA was tried to determine whether this would result in better class-separability. For every datapoint in the full dataset, the classes of the group of ten most similar datapoints (based on cosine similarity) were determined. By maximizing the average of the number of within-group same-class datapoints, the used dimensionality was determined.

3   Results

This section will go onto visualize and describe the achieved results for all three experiments described in Section 2.5. Note that for all figures, the results were averaged over three runs with the error bars showing one standard deviation. Furthermore, all deficiencies were rounded to two decimal places. For deficiency values <1 (improvements over the reference strategy), we show the smallest value in the comparison in bold. For the sake of readability and to keep graph points aligned, in the graphs for query-pool sizes of 0.5% and 1% the points shown are respectively those at every 10th and 5th and interval.

3.1   Active Learning

Figure 1a shows how the query functions performed on the KvK dataset. All query functions outperform random sampling when the labeled dataset is less than 200 examples large. After this, in particular BALD and variation ratio continue to mostly outperform random sampling until near the maximum labeled data size. Notably, many of the performance differences are larger than one standard deviation.

Figure 1b shows how random sampling and the implemented query functions performed on the SST dataset. On this dataset the results for the random sampling baseline and the other systems is much smaller, and there does not seem to be a clear winner.
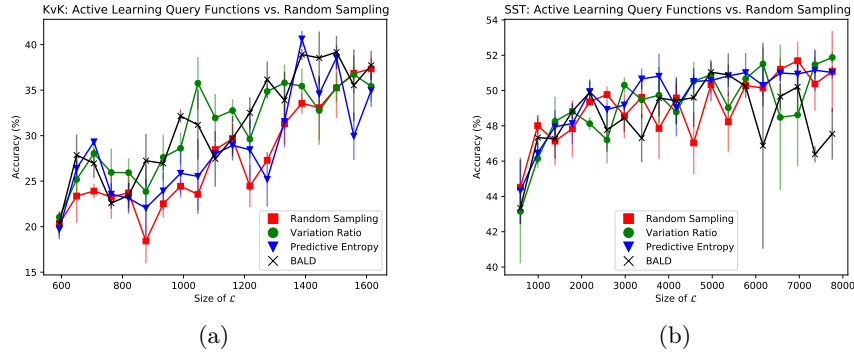
Fig. 1. The achieved test accuracy on the KvK dataset (a) and on the SST dataset (b) by random sampling and the uncertainty-based query functions.

Finally, the deficiencies shown in Table 5 show a positive result (< 1) for all query functions except for predictive entropy for the SST dataset. Matching the graphs, the performance gains as measured by the deficiency scores are overall more substantial on the KvK dataset. BALD has the lowest deficiency for both datasets.

Table 5. The deficiencies (Eq. 2) of the uncertainty-based query functions. Random sampling was the reference strategy.

Dataset	VR	PE	BALD
SST	0.95	1.01	**0.89**
KvK	0.67	0.9	**0.64**

3.2   Query-pool Size

Figure 2a shows the performance of variation ratio across different q when used on the KvK dataset. In the middle range of the graph, variation ratio with a q of 5% has a worse performance than the other q. The q of 0.5% and 1% achieve similar performance with the accuracy scores always staying within one standard deviation of each other.

Figure 2b shows the performance of the different q on the SST dataset. The performance of variation ratio with a q of 0.5% fluctuates more when compared to the other q. Moreover, it results in an overall worse performance when compared to the other sizes. The q of 5% shows to have the best and most consistent performance over the whole learning curve in terms accuracy. However, the q of 0.5% manages to outperform the other q at about 5000 labeled examples.
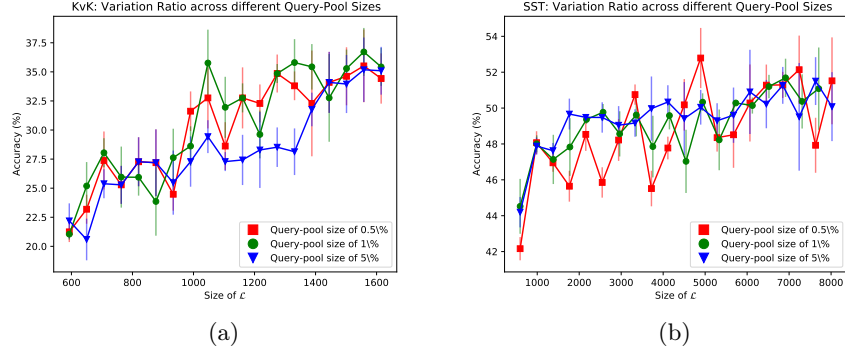
Fig. 2. The achieved test accuracy on the KvK dataset (a) and the SST dataset (b) by using the variation ratio query function with different q.

The deficiencies for the different q across both datasets are shown in Table 6. For the SST dataset, the q of 5% had a lower deficiency across the learning curve whereas the q of 0.5% shows a relatively high deficiency. For the KvK dataset however, we see that the q of 5% has a relatively high deficiency when compared to the similarly performing q of 0.5% and 1%.

Table 6. The achieved deficiencies (Eq. 2) by the different q for the different datasets. A q of 1% was the reference strategy.

Dataset	0.5%	5%
SST	1.65	**0.62**
KvK	**0.91**	1.33

3.3   Heuristics

Figure 3a shows the performance of using variation ratio with heuristics together with the performance of solely using variation ratio on the KvK dataset (also shown in Figure 1b). Both RET and RECT show no clear improvement over solely using variation ratio. The same can be gathered from the results of the SST dataset shown in Figure 3b as their accuracy scores stay within one standard deviation for the entire learning curve. Moreover, Table 7 shows that the average Kendall's τ is around 0 with a relatively large standard deviation; indicating that there is no relationship between the compared rankings.

Lastly, SUD shows an overall worse performance for both the SST and KvK datasets. The deficiencies shown in Table 8 also show high values for SUD across both datasets.
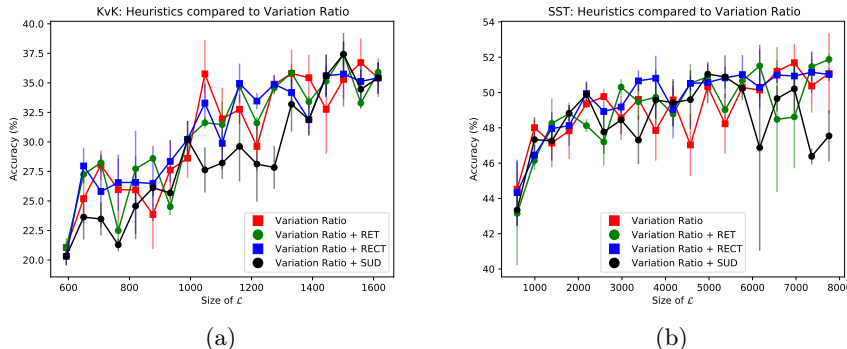
(a)   (b)

Fig. 3. The achieved test accuracy on the KvK dataset (a) and on the SST dataset (b) by the different heuristics.

Table 7. The mean and the 1 SD range of Kendall's τ from the described ranking experiment across the two datasets (rounded to two decimal places).

Dataset	Mean	σ
SST	0.14	0.33
KvK	0.02	0.47

Table 8. The achieved deficiencies by the different heuristics. Variation ratio was the reference strategy.

Dataset	RET	RECT	SUD
SST	1.02	1.05	1.23
KvK	0.98	**0.96**	1.33

4   Discussion

This research investigated whether AL could be used to reduce labeling effort while at the same time maintaining similar performance to a model trained on a full dataset. To achieve this, the performance and scalability of different AL query-strategies was tested for the state-of-the-art NLP model: BERT.

Conclusions   The results showed that uncertainty-based AL can provide improved performance over random sampling for cut-down datasets. This difference was not consistent throughout the whole training curve: at specific points AL outperformed random sampling and at others at it achieved similar performance. Unfortunately, the results found for the KvK dataset show that the found improvement can diminish as query-pool sizes get larger. Moreover, the two proposed heuristics aimed at improving scalability did not help in improving performance for either dataset and the heuristic aimed at avoiding outliers even resulted in worse performance. An unexpected result was found in that the assumption that semantically similar data conveyed the same type of information did not hold according to the conducted ranking experiment. A possible explanation for this could be that the texts were not mapped to embeddings in a way in which semantically similar data was close enough to each other. Another

curious finding was that for the SST dataset, the smallest q resulted in the worse performance, especially at the beginning of the learning curve. This is counter-intuitive due to the fact that performance seems to suffer from more frequent uncertainty estimates. Moreover, for the KvK dataset we did see that the largest q had the worst performance overall but also had similar performance to the other q. A potential justification for this could be that updating too frequently at the beginning of the learning curve results in the model not being able to train enough on high frequency classes and focuses too much on the long tail of the class distribution due to the fact that the model is more uncertain about texts with low frequency classes at the start of the learning curve. Further research is needed to build a better understanding of this.

From the above, we conclude that uncertainty-based AL with BERT$_{base}$ can be used to decrease labeling effort. This supports what was concluded by [10].

When looking at the bigger picture, we showed that AL can still provide an improvement in performance over random sampling for large datasets. The improvement of performance of AL with BERT is however limited when compared to what it achieved for older NLP models [34,30,24] and even more so when compared to image classifiers [14,5,9]. Performance did show to increase more when used on the smaller dataset. A possible explanation for this is that BERT is pretrained on a large amount of data and that it only needs fine-tuning for achieving good performance on a specific task. Transfer learning models [13] like BERT have the ability to perform well on new tasks with just a limited amount of data. The power of this few-shot learning also became apparent on a dataset which we decided not to use. Here, BERT was able to get a low validation error on the seed alone, while at the same time having a training accuracy of 100%.

Future Research This work focused on classification tasks. A future direction could be to investigate the influence of AL on BERT's performance in the context of regression tasks and also to examine how the proposed heuristics perform there. Moreover, more recent BERT variants, like for instance RoBERTa [19], could be tested to see whether AL still outperforms the random sampling benchmark. Furthermore, the used query functions were mostly developed for and used in computer vision. Query functions aimed at text classification or at the fact that BERT is a pretrained model could be further investigated.

Lastly, an important direction for future work remains making AL more scalable, by finding ways to preserve performance with larger query-pool sizes.

Acknowledgments

References

1. Ahmed, W., Natarajan, T., Rao, K.R.: Discrete Cosine Transform. IEEE Transactions on Computers **23**(1), 90–93 (1974)
2. Bouneffouf, D., Laroche, R., Urvoy, T., Féraud, R., Allesiardo, R.: Contextual Bandit for Active Learning: Active Thompson Sampling. In: Proceedings of the 21st International Conference on Neural Information Processing (ICONIP). pp. 405–412 (2014)
3. Cai, W., Zhang, Y., Zhou, J.: Maximizing Expected Model Change for Active Learning in Regression. In: Proceedings - IEEE International Conference on Data Mining, ICDM. pp. 51–60 (December 2013)
4. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. In: North American Association for Computational Linguistics (NAACL). pp. 4171–4186 (2019)
5. Drost, F.: Uncertainty Estimation in Deep Neural Networks for Image Classification. Master's thesis, University of Groningen (2020)
6. Ein-Dor, L., Halfon, A., Gera, A., Shnarch, E., Dankin, L., Choshen, L., Danilevsky, M., Aharonov, R., Katz, Y., Slonim, N.: Active Learning for BERT: An Empirical Study. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 7949–7962 (2020)
7. Gal, Y.: Uncertainty in Deep Learning. Master's thesis, University of Cambridge (2016)
8. Gal, Y., Ghahramani, Z.: Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In: Proceedings of The 33rd International Conference on Machine Learning. vol. 48, pp. 1050–1059. PMLR (2016)
9. Gal, Y., Islam, R., Ghahramani, Z.: Deep bayesian active learning with image data. In: Proceedings of the 34th International Conference on Machine Learning. vol. 70, pp. 1183–1192. PMLR (Augustus 2017)
10. Grießhaber, D., Maucher, J., Vu, N.T.: Fine-tuning BERT for Low-Resource Natural Language Understanding via Active Learning. CoRR **abs/2012.02462** (2020)
11. Gulati, P., Sharma, A., Gupta, M.: Theoretical Study of Decision Tree Algorithms to Identify Pivotal Factors for Performance Improvement: A Review. International Journal of Computer Applications **141**(14), 19–25 (2016)
12. Guo, C., Pleiss, G., Sun, Y., Weinberger, K.Q.: On calibration of modern neural networks. In: International Conference on Machine Learning. pp. 1321–1330 (2017)
13. Gupta, A., Thadani, K., O'Hare, N.: Effective Few-Shot Classification with Transfer Learning. In: Proceedings of the 28th International Conference on Computational Linguistics. pp. 1061–1066 (December 2020)
14. Houlsby, N., Huszár, F., Ghahramani, Z., Lengyel, M.: Bayesian active learning for classification and preference learning (2011)
15. Hu, P., Lipton, Z.C., Anandkumar, A., Ramanan, D.: Active Learning with Partial Feedback. CoRR **abs/1802.07427** (2018)
16. Kingma, D., Ba, J.: Adam: A Method for Stochastic Optimization. CoRR **abs/1412.6980** (2015)
17. Krogh, A., Vedelsby, J.: Neural Network Ensembles, Cross Validation and Active Learning. In: Proceedings of the 7th International Conference on Neural Information Processing Systems. pp. 231–238. MIT Press (1994)
18. Kullback, S., Leibler, R.A.: On Information and Sufficiency. The Annals of Mathematical Statistics **22**(1), 79–86 (1951)

19. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: RoBERTa: A Robustly Optimized BERT Pretraining Approach. CoRR **abs/1907.11692** (2019)
20. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient Estimation of Word Representations in Vector Space. In: Proceedings of Workshop at ICLR. pp. 1–12 (January 2013)
21. Oosten, J.P., Schomaker, L.: Separability versus prototypicality in handwritten word-image retrieval. Pattern Recognition **47**(3), 1031–1038 (2014)
22. Pennington, J., Socher, R., Manning, C.: GloVe: Global Vectors for Word Representation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 1532–1543 (October 2014)
23. Reimers, N., Gurevych, I.: Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. CoRR **abs/1908.10084** (2019)
24. Roy, N., McCallum, A.: Toward optimal active learning through sampling estimation of error reduction. In: Proceedings of the Eighteenth International Conference on Machine Learning. pp. 441—448 (August 2001)
25. Settles, B.: Active Learning Literature Survey. Synthesis Lectures on Artificial Intelligence and Machine Learning **6**(1), 1–114 (2012)
26. Shannon, C.E.: A mathematical theory of communication. Bell System Technical Journal **27**(3), 379–423 (1948)
27. Smith, L.N.: No More Pesky Learning Rate Guessing Games. CoRR **abs/1506.01186** (2015)
28. Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C.D., Ng, A., Potts, C.: Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In: Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing. pp. 1631–1642 (October 2013)
29. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research **15**(56), 1929–1958 (2014)
30. Tang, M., Luo, X., Roukos, S.: Active learning for statistical natural language parsing. In: Proceedings of ACL 2002. pp. 120–127 (July 2002)
31. Teye, M., Azizpour, H., Smith, K.: Bayesian Uncertainty Estimation for Batch Normalized Deep Networks. In: Proceedings of the 35th International Conference on Machine Learning. vol. 80, pp. 4907–4916. PMLR (July 2018)
32. Tsymbalov, E., Panov, M., Shapeev, A.: Dropout-Based Active Learning for Regression. Analysis of Images, Social Networks and Texts pp. 247–258 (2018)
33. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems. pp. 6000—-6010 (2017)
34. Zhu, J., Wang, H., Yao, T., Tsou, B.K.: Active learning with sampling by uncertainty and density for word sense disambiguation and text classification. In: Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008). pp. 1137–1144 (2008)

Appendix

A.1   Query Functions

Variation Ratio The variation ratio is a measure of dispersion around the class that the model predicts most often (the mode). The intuition here is that the model is uncertain about a datapoint when it has predicted the mode class a relatively small number of times. This indicates that it has predicted other classes a relatively large number of times. This can also be gathered from Equation 3, where f_x denotes the mode count and T the number of stochastic forward passes.

$$v[x] = 1 - \frac{f_x}{T} \tag{3}$$

The function attains its maximum value when the model predicts all classes an equal amount of times and its minimum value when the model only predicts one class across all stochastic forward passes. Variation ratio only captures the uncertainty contained in the predictions, not the model, as it only takes into account the spread around the most predicted class. It is thus a form of predictive uncertainty.

Predictive Entropy Entropy $H(x)$ in the context of information theory is defined as:

$$H(x) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{4}$$

This formula expresses the entropy in bits per symbol to be communicated, in which $p(x_i)$ gives the probability of the i-th possible value for the symbol. Entropy is used to quantify the information of data. In our case we want to know the chance of the model classifying a datapoint as a certain class given the input and model parameters ($p(y = c|\mathbf{x}, \boldsymbol{\omega})$). We can compute this chance by averaging over the softmax probability distributions across the T stochastic forward passes. This adjusted version of entropy is denoted in Equation 5, where $\hat{\omega}_t$ denotes the stochastic forward pass t, and c the number associated to the class-label.

$$\begin{aligned} H[y|\mathbf{x}, \mathcal{D}_{train}] = -\sum_c & \left(\frac{1}{T} \sum_t p(y = c|\mathbf{x}, \hat{\boldsymbol{\omega}}_t) \right) \\ & \log \left(\frac{1}{T} \sum_t p(y = c|\mathbf{x}, \hat{\boldsymbol{\omega}}_t) \right) \end{aligned} \tag{5}$$

To exemplify: in binary classification, the predictive entropy is highest when the model its softmax classifications consist of T times [0.5, 0.5]. In that case, expected surprise when we would come to know the real class-label is at its highest. The uncertainty is computed averaging over all predictions and thus falls under predictive uncertainty.

Bayesian Active Learning by Disagreement Predictive entropy (Section A.1) is used to quantify the information in one variable. Mutual information or joint entropy is very similar but is used to calculate the amount of information one variable conveys about another. In our case, we'll be looking at what the average model prediction will convey about the model posterior, given the training data. This is a form of conditional mutual information, the condition or the third variable being the training data $\mathcal{D}_{train}$. [14] used this form of mutual information in an AL setting and dubbed it Bayesian active learning by disagreement (BALD).

$$
\begin{aligned}
I[y, \omega | \mathbf{x}, \mathcal{D}_{train}] = -\sum_c & \left(\frac{1}{T} \sum_t p(y = c | \mathbf{x}, \hat{\boldsymbol{\omega}}_t) \right) \\
& \log \left(\frac{1}{T} \sum_t p(y = c | \mathbf{x}, \hat{\boldsymbol{\omega}}_t) \right) \\
& -\frac{1}{T} \sum_{c,t} p(y = c | \mathbf{x}, \hat{\boldsymbol{\omega}}_t) \\
& \log p(y = c | \mathbf{x}, \hat{\boldsymbol{\omega}}_t)
\end{aligned}
\tag{6}
$$

The difference between Equation 6 and 5 is that the conditional entropy is subtracted from the predictive entropy. The conditional entropy is the probability of the full output being generated from the training data and the input. This is the reason we do not average the predictions for every single class. We first sum over all classes, so that we do not average over the model parameters for every single class and thus take into account the fact that we are looking at the chance of the complete probability distribution being generated.

BALD is maximized when the T predictions are strongly disagreeing about what label to assign to the example. So in the binary case, it would be highest when the predictions would alter between [1,0] and [0,1] as these two predictions are each others complete opposite. Unlike the variation ratio and predictive entropy, BALD is a form of model uncertainty. When the softmax outputs would be equal to T times [0.5,0.5], the minimal BALD value would be returned as the predictions are the same and the model is thus very confident about its prediction.

A.2   RET Algorithm Computational Cost Analysis

The number of forward passes required by the RET algorithm depends on two factors:

1. *Basic passes*: The forward passes required by the "normal" computation of uncertainty at the beginning of the computation for every query-pool.
2. *RP passes*: The forward passed required for intermediate updates, using the redundancy pool RP.

In this analysis we will assume that the size of the redundancy pool $|\mathcal{RP}|$ is chosen as a factor $f > 1$ of the size of the query-pool q. A reasonable assumption, considering that making $|\mathcal{RP}|$ larger than needed incurs unnecessary computational cost, whereas a too small value is expected to diminish the effect of the RET algorithm. We furthermore notice that given this assumption, and assuming a fixed total number of examples to label, there are two factors influencing the required amount of *RP passes*:

– Linearly increasing the query-pool size and coupled redundancy pool size causes a quadratic increase in the number of required forward passes per query pool round.
– At the same time, a linearly increased query-pool size also induces a corresponding linear decrease in the number of required query-pool rounds.

We will see that these two factors will cause a net linear contribution to the number of *RP passes* starts causing a net increase of total passes once the query-size comes above a certain value. Looking at (1) more precisely, the amount of passes over $\mathcal{RP}$ that needs to be performed per query-pool round can be computed as an *arithmetic progression*:

$$|\mathcal{RP}| + (|\mathcal{RP}| - 1) + (|\mathcal{RP}| - 2) + \ldots + (|\mathcal{RP} - q) = \tag{7}$$

$$\frac{1}{2} \times (q + 1) \times (|\mathcal{RP}| + |\mathcal{RP}| - q) = \tag{8}$$

$$\frac{1}{2} \times (q + 1) \times ((2f - 1) \times q) = \tag{9}$$

$$\frac{1}{2} \times (q + 1) \times f' \times q = \tag{10}$$

$$\frac{1}{2} \times f' \times (q^2 + q)) \tag{11}$$

Let's assume we use $f = 1.5$ (as also used in our experiments), and consequently, $f' = 2f - 1 = 2$. The number of forward passes over $\mathcal{RP}$ then becomes exactly $q^2 + q$.

The complexity can then be expressed by the following formula:

$$T \times \lceil \frac{\#\text{Samples}}{q} \rceil \times (|\text{data}| + q^2 + q) \tag{12}$$

This can be approximately rewritten as:

$$T \times \#\text{Samples} \times (\frac{|\text{data}|}{q} + \frac{q^2 + q}{\text{query-pool}}) = \tag{13}$$

$$T \times \#\text{Samples} \times (\frac{|\text{data}|}{q} + q + 1) \tag{14}$$

Note that the second term query-pool-size $+ 1$ only starts dominating the number of forward passes in this formula as soon as:

$$q + 1 \approx q > \frac{|\text{data}|}{q}$$

This is the case when

$$q > \sqrt{(|\text{data}|)}$$

Until then, the computational gains of less *basic passes* outweighs the cost of more *RP passes*. In practice though, this may happen fairly quickly. For example, assuming we have a data size of 10000 examples, and we use as mentioned $q = 1.5| \times \mathcal{RP}|$, then as soon as $q \geq 100$ the increased computation of the *RP passes* starts dominating the gains made by less *basic passes* when further increasing the query-pool size, and the net effect is that the total amount of computation increases.

In summary, for the RET algorithm, *RP passes* contribute to the total amount of forward passes. Furthermore, this contribution increases linearly with redundancy-pool size and coupled query-pool size, and starts to dominate the total amount of forward passes once redundancy-pool-size $> \sqrt{\text{data-size}}$. This limits its use for decreasing computation by increasing the query-pool size.

A.3   Algorithms

Algorithm 1 The general AL loop.

Input Labeled dataset $\mathcal{L} = \{(x_i, y_i)\}_i^n$, the unlabeled data $\mathcal{U} = \{(x_i, \emptyset)\}_i^n$ and the untrained classifier $f(x; \theta)$.

Output Fully labeled dataset $\mathcal{L} = \{(x_i, y_i)\}_i^n$ and trained classifier $f(x; \theta)$

1: $n \leftarrow$ Desired length of $\mathcal{L}$
2: $q \leftarrow$ Query-pool size
3: $Q(x) \leftarrow$ Query Function
4: **while** $\mathcal{L}$ length $< n$ **do**
5:     Retrain $f(x; \theta)$ on $\mathcal{L}$
6:     Sort $\mathcal{U}$ based on $Q(\mathcal{U})$
7:     Let Oracle assign labels to $\mathcal{U}_0^q$
8:     Insert $\mathcal{U}_0^q$ into $\mathcal{L}$
9:     Remove $\mathcal{U}_0^q$ from $\mathcal{U}$
10: **end while**

Algorithm 2 The AL loop with MCD.

Input Labeled dataset $\mathcal{L} = \{(x_i, y_i)\}_i^n$, the unlabeled data $\mathcal{U} = \{(x_i, \emptyset)\}_i^n$ and the untrained classifier $f(x; \theta)$.

Output Fully labeled dataset $\mathcal{L} = \{(x_i, y_i)\}_i^n$ and trained classifier $f(x; \theta)$

1: $n \leftarrow$ Desired dataset length
2: $q \leftarrow$ Query-pool size
3: $Q(x) \leftarrow$ Query Function
4: $T \leftarrow$ Number of SFP's
5: **while** $\mathcal{L}$ length $< n$ **do**
6:     Retrain $f(x; \theta)$ on $\mathcal{L}$
7:     $P \leftarrow \emptyset$
8:     **for** $t = 0, ..., T$ **do**
9:         insert $f(\mathcal{U}; \theta_t)$ into P
10:     **end for**
11:     Sort $\mathcal{U}$ based on $Q(P)$
12:     Let Oracle assign labels to $\mathcal{U}_0^q$
13:     Insert $\mathcal{U}_0^q$ into $\mathcal{L}$
14:     Remove $\mathcal{U}_0^q$ from $\mathcal{U}$
15: **end while**

Algorithm 3 The AL loop with Redundancy Elimination by Training (RET).

Input Labeled dataset $\mathcal{L} = \{(x_i, y_i)\}_i^n$, the unlabeled data $\mathcal{U} = \{(x_i, \emptyset)\}_i^n$ and the untrained classifier $f(x; \theta)$.
Output Fully labeled dataset $\mathcal{L} = \{(x_i, y_i)\}_i^n$ and trained classifier $f(x; \theta)$

1: $n \leftarrow$ Desired dataset length
2: $r \leftarrow$ Redundancy-pool size
3: $q \leftarrow$ Query-pool size
4: $T \leftarrow$ Number of SFP's
5: $Q(x) \leftarrow$ Query Function
6: **while** $\mathcal{L}$ length $< n$ **do**
7:     Retrain $f(x; \theta)$ on $\mathcal{L}$
8:     $P \leftarrow \emptyset$
9:     **for** $t = 0, ..., T$ **do**
10:        insert $f(\mathcal{U}; \theta_t)$ into P
11:     **end for**
12:     Sort $\mathcal{U}$ based on $Q(P)$
13:     $U \leftarrow \emptyset$
14:     $queried \leftarrow 0$
15:     **while** $queried < q$ **do**
16:         **for** $t = 0, ..., T$ **do**
17:             insert $f(\mathcal{RP}; \theta_t)$ into U
18:         **end for**
19:         $i \leftarrow argmin(U)$
20:         Let Oracle assign label to $\mathcal{U}_i$
21:         Train $f(x; \theta)$ on $\mathcal{U}_i$
22:         Insert $\mathcal{U}_i$ into $\mathcal{L}$
23:         Remove $\mathcal{U}_i$ from $\mathcal{U}$
24:         $queried \leftarrow queried + 1$
25:     **end while**
26: **end while**

Algorithm 4 The AL loop with Redundancy Elimination by Cosine Similarity (RECS).

Input Labeled dataset $\mathcal{L} = \{(x_i, y_i)\}_i^n$, the unlabeled data $\mathcal{U} = \{(x_i, \emptyset)\}_i^n$ and the untrained classifier $f(x; \theta)$.

Output Fully labeled dataset $\mathcal{L} = \{(x_i, y_i)\}_i^n$ and trained classifier $f(x; \theta)$

1: $n \leftarrow$ Desired dataset length
2: $u \leftarrow$ Redundancy-pool size
3: $q \leftarrow$ Query-pool size
4: $l \leftarrow$ Cosine similarity threshold
5: $T \leftarrow$ Number of SFP's
6: $Q(x) \leftarrow$ Query Function
7: $Cos(x, y) \leftarrow$ Cosine similarity between x and y
8: **while** $\mathcal{L}$ length $< n$ **do**
9:     Retrain $f(x; \theta)$ on $\mathcal{L}$
10:     $P \leftarrow \emptyset$
11:     **for** $t = 0, ..., T$ **do**
12:         insert $f(\mathcal{U}; \theta_t)$ into P
13:     **end for**
14:     Sort $\mathcal{U}$ based on $Q(P)$
15:     $U \leftarrow \emptyset$
16:     **while** $U length < q$ **do**
17:         **for** $i = 0, ..., u$ **do**
18:             **if** $Cos(\mathcal{U}_i, U_0^{Ulength}) < l$ **then**
19:                 insert $\mathcal{U}_i$ into U
20:             **end if**
21:         **end for**
22:         $l \leftarrow l - 0.01$
23:     **end while**
24:     Reset l to initial value
25:     Let Oracle assign labels to U
26:     Insert U into $\mathcal{L}$
27:     Remove U from $\mathcal{U}$
28: **end while**

Algorithm 5 The AL loop with SUD.

Input Labeled dataset $\mathcal{L} = \{(x_i, y_i)\}_i^n$, the unlabeled data $\mathcal{U} = \{(x_i, \emptyset)\}_i^n$ and the untrained classifier $f(x; \theta)$.

Output Fully labeled dataset $\mathcal{L} = \{(x_i, y_i)\}_i^n$ and trained classifier $f(x; \theta)$

1: $n \leftarrow$ Desired dataset length
2: $q \leftarrow$ Query-pool size
3: $k \leftarrow$ Amount of similar examples to compute density with
4: $T \leftarrow$ Number of SFP's
5: $Q(x) \leftarrow$ Query Function
6: $Cos(x, y) \leftarrow$ Cosine similarity between x and y
7: **while** $\mathcal{L}$ length $< n$ **do**
8:     Retrain $f(x; \theta)$ on $\mathcal{L}$
9:     $P \leftarrow \emptyset$
10:    $E \leftarrow \emptyset$
11:    **for** $t = 0, ..., T$ **do**
12:        Insert $f(\mathcal{U}; \theta_t)$ into P
13:    **end for**
14:    **for** $example$ in $\mathcal{U}$ **do**
15:        $similar \leftarrow Sort(Cos(example, U))$
16:        Insert $\frac{sum(similar_0^k))}{k}$ into E
17:    **end for**
18:    Sort $\mathcal{U}$ based on $Q(P * E)$
19:    Let Oracle assign labels to $\mathcal{U}_0^q$
20:    Insert $\mathcal{U}_0^q$ into $\mathcal{L}$
21:    Remove $\mathcal{U}_0^q$ from $\mathcal{U}$
22: **end while**