# Active learning and its applications for reducing labeling effort in text classification tasks

Bachelor's Project Thesis

Pieter Floris Jacobs, s3777693, p.f.jacobs@student.rug.nl,
Supervisors: dr. M.A. Wiering & dr. G. Maillette de Buy Wenninger

**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. Our results show that using uncertainty-based AL with BERT$_{base}$ outperforms randomly sampling data. This difference in performance can decrease as the query-pool size gets larger. It was also found that the proposed heuristics did not significantly improve performance when compared to the basic implementation of uncertainty-based AL.

## 1 Introduction

Deep learning is a field in machine learning in which neural networks with a large number of layers (therefore called deep networks) 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 research will focus on an instance of supervised learning, called text classification. An example of mapping a text to a certain output would be classifying a text on its quality, where the labels could for instance be {low, medium, high}. Data labeling is usually done manually and can grow to be an expensive and time-consuming task for larger datasets, like those used in deep learning. This begs the question of whether there is no way to reduce the labeling effort while still being able to reach an acceptable performance on the network modeling the chosen task. Similarly to lossy compression (Ahmed, Natarajan, & Rao, 1974), we want to retain a good approximation of the original dataset while at the same time reducing its size as much as possible. Or to put it more specifically: given a training set, how can we optimally choose a limited number of examples based on the amount of relevant info they contain for the target task?

To be able to determine what data contains the most information, we need to quantify the amount of information contained in a datapoint. This finds its roots, like lossy compression, in information theory. Shannon (1948) introduced informational entropy, for which the definition can be found in Equation 1.1 and where $p(x_i)$ is the probability of $x_i$. It is a measure of the number of binary digits (bits) required for encoding a message. This number of bits is in turn a measure of the amount of information contained in a message.

$$H(x) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \qquad (1.1)$$

A model trained on limited data has an entropy associated with its target variable predictions. Our goal is to select the data that is to be labeled in a greedy way to reduce this entropy as much as possible, similar to how it is done in research on decision trees (Gulati, Sharma, & Gupta, 2016). In essence, we are trying to maximize the information gain or minimize the Kullback-Leiber divergence (Kullback & Leibler, 1951). To exemplify: this is quite similar to a game of 'Guess Who?'. We are trying to determine what person we are with as little guesses as possible. Moreover, in 'Guess Who?' you change your strategy based on the answers you get from your opponent. This is akin to a problem that we have to overcome in the field of deep learning, albeit more troublesome there. This problem being that the parameters of a neural network change during training, and therefore its predictions and certainty of new datapoints will also change. This makes it so that we cannot calculate the entropy for all datapoints in one go to then split the dataset in a way that results in the highest information gain.

A machine learning technique called Active Learning (AL) (Settles, 2012) can be used to combat this problem. In AL, a human labeler is queried for datapoints that the network itself deems most informative with its current parameter configuration. The human labeler proceeds to assign the right label to these queried datapoints and then the network is retrained on them. This process is repeated until the model shows robust performance, which in turn is an indication 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 highest model change (Cai, Zhang, & Zhou, 2013) or through treating the model as a multi-arm bandit (Bouneffouf, Laroche, Urvoy, Féraud, & Allesiardo, 2014). Most of the existing literature on this does this through different measures of model uncertainty though (Drost, 2020; Gal, 2016; Gal & Ghahramani, 2016; Gal, Islam, & Ghahramani, 2017; Teye, Azizpour, & Smith, 2018) and this is also what this research will focus on. Bayesian probability theory provides us with the necessary mathematical tools to reason about uncertainty, but in the space of deep learning it has its complications.

To clarify this, it is essential to note that classification neural networks are disciminative systems. A classifier is trained to map one or more inputs to one or more outputs, or to put it more formal: to learning a certain $f(x) = y$. After the classifier has learned such a function and the training process is over, its parameters do not change anymore. This in turn means that the same input will from then on always map to the same output, or in our case the same text to the same label.

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* as seen in Equation 1.2. Here is where the issue of the classifier being discriminative becomes apparent. Namely, that there is no probability distribution over what output gets generated from a certain input: the output is always the same for a given input. A generative model is needed to be able to reason about the relation between input and output in this manner. What is more, even if we suppose the network was generative the integral would not be analytically solvable due to the fact that we would need to integrate over all possible parameter settings $\omega$.

$$p(y * | x*, X, Y) = \int p(y * | x*, \omega) p(\omega, X, Y) d\omega \tag{1.2}$$

However, it can be approximated. Existing literature has explored different methods of achieving this, with Monte Carlo Dropout (MCD) being the most popular one (Drost, 2020; Gal & Ghahramani, 2016; Tsymbalov, Panov, & Shapeev, 2018). In MCD, the network makes use of dropout (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014) 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 summarise the uncertainty of the model in a variety of ways. A more extensive description of MCD can be found in Section 2.2.2.

By making use of the MCD approximation, this research will go on to focus on comparing different uncertainty-related AL query methods for text classification, a space in which there is still little literature on the usability of AL for modern NLP models. We will try 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 prevalence of transfer learning and scalability have to be taken into account. The goal of this being to reach a performance similar to the state-of-the-art text-classification models that use random sampling on a complete training set. This should provide insight into 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 the finally the experimental setup.

## 2.1 Active Learning

An implementation of the general AL loop/round is shown in the Appendix (Algorithm A.1). It consists of four separable 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 (Hu, Lipton, Anandkumar, & Ramanan, 2018).

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.

Note the datasets used for the experiments (Section 2.5.1) 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 when 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

### 2.2.1 BERT

The model used to classify the texts was BERT$_{base}$ (Devlin, Chang, Lee, & Toutanova, 2019), which is a variant of the Transformer model (Vaswani et al., 2017). This model was chosen due to the fact that BERT is currently considered the state-of-the-art language model.

BERT$_{base}$ has its own tokenizer based on the WordPiece tokenization method (Wu et al., 2016). This tokenizer has a cased and an uncased version. The uncased version was used in this research, as the information of capitalization and accent markers preserved in the cased version was not deemed to be of significant importance for the used tasks and datasets. 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. This choice had to be made due to computational constraints and is discussed in Section 4.2.

Furthermore, the option of the BERT$_{base}$ tokenizer to make use of special tokens for sentence seperation, padding, masking and to generalise unknown vocabulary was used. This was done as it was assumed to provide BERT$_{base}$ with a better understanding of the texts, considering that our data occasionally contained unknown vocabulary and shorter texts which needed padding.

Finally, a softmax layer was added to the end of BERT$_{base}$. Equation 2.1 shows how the softmax probability for input vector $\mathbf{z}$ is computed.

$$\sigma(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp z_j} \text{for } i = 1, ..., K. \tag{2.1}$$

Applying softmax to an input vector normalizes that vector into a probability distribution that sums to one. This softmax layer is essential as the implemented query functions (Section 2.3) compute uncertainty based on sampled output probability distributions*.

### 2.2.2 Monte Carlo Dropout

Monte Carlo dropout (MCDO) is, like discussed in Section 1, a technique that enables reasoning about uncertainty with neural networks. Dropout (Srivastava et al., 2014) essentially 'turns off' neurons with a predefined probability. Every forward pass, the neurons have a predefined chance to not be used in the computation of the output. 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 (Gal & Ghahramani, 2016) through creating $T$ predictions for all datapoints. To emphasise: every datapoint will now have $T$ different predictions associated with it, as $T$ slightly altered models will be used on the same datapoint. 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 A.2).

BERT$_{base}$ has two different types of dropout layers: hidden dropout and attention dropout. The hidden dropout is the dropout for all fully connected layers in the embedding, the encoder, and the pooler. The attention dropout applies to all attention layers in the multi-head attention block. 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 (Krogh & Vedelsby, 1994)
- Monte Carlo Batch Normalization (MCBN) (Teye et al., 2018).

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 (Drost, 2020).

*Active learning using dropout-based uncertainty estimation is also possible for regression, but requires an adapted approach to compute uncertainty and associated query functions, see for example Tsymbalov et al. (2018)

### 2.2.3 SentenceBERT

Textual data offers the advantage of having access to the use of embeddings. A word embedding is a learned representation of word into a vector space in which semantically similar words are close to each other. However, text classification involves classifying texts made up of one or multiple sentences. Fortunately, these 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 (Pennington, Socher, & Manning, 2014) and averaging embeddings created by a Word2Vec model (Mikolov, Chen, Corrado, & Dean, 2013). We have opted to make use of SentenceBERT (Reimers & Gurevych, 2019), a siamese BERT architecture which has shown to have better performance than the previously mentioned strategies. To clarify: SentenceBERT was used seperately from the previously discussed BERT$_{base}$ model and was used for the sole use of assigning embeddings to each sentence in the dataset that were used by the heuristics described in Sections 2.4.1 and 2.4.2.

## 2.3 Query Functions

The query function is the way by which the model chooses what data it wants to learn from. It is central to the AL loop and, as previously mentioned in Section 1, this paper will focus on query functions that reason about uncertainty. All used query functions make use of $T$ stochastic forward passes (Section 2.2.2). In our case, the result of these stochastic forward passes will be $T$ softmax probability distributions for every datapoint. These are essential to reason about uncertainty, as they provide us with an approximated Bayesian distribution for every datapoint (Gal & Ghahramani, 2016).

The three functions that were implemented were the variation ratio, predictive entropy, and Bayesian active learning by disagreement. They all compute uncertainty differently, and their differences will be highlighted. For a more extensive discussion of Equations 2.2, 2.3 and 2.4 one is encouraged to look at Gal (2016).

### 2.3.1 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 2.2, where $f_x$ denotes the mode count and $T$ the number of stochastic forward passes.

$$v[x] = 1 - \frac{f_x}{T} \tag{2.2}$$

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.

### 2.3.2 Predictive Entropy

As discussed in Section 1, entropy (Equation 1.1) 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 2.3, where $\hat{\omega_t}$ denotes the stochastic forward pass $t$, and $c$ the number associated to the class-label.

$$\begin{aligned} H[y|\mathbf{x}, \mathcal{D}_{train}] = -\sum_c \left(\frac{1}{T}\sum_t p(y = c|\mathbf{x}, \hat{\boldsymbol{\omega}}_{\boldsymbol{t}})\right) \\ \log\left(\frac{1}{T}\sum_t p(y = c|\mathbf{x}, \hat{\boldsymbol{\omega}}_{\boldsymbol{t}})\right) \end{aligned} \tag{2.3}$$

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.

### 2.3.3 Bayesian Active Learning by Disagreement

Predictive entropy (Section 2.3.2) 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 $\mathcal{D}_{train}$. Houlsby, Huszár, Ghahramani, and Lengyel (2011) used this form of mutual information in an AL setting and dubbed it Bayesian active learning by disagreement (BALD).

$$\begin{aligned} I[y, \omega|\mathbf{x}, \mathcal{D}_{train}] = -\sum_c \left(\frac{1}{T}\sum_t p(y = c|\mathbf{x}, \hat{\boldsymbol{\omega}}_{\boldsymbol{t}})\right) \\ \log\left(\frac{1}{T}\sum_t p(y = c|\mathbf{x}, \hat{\boldsymbol{\omega}}_{\boldsymbol{t}})\right) \\ -\frac{1}{T}\sum_{c,t} p(y = c|\mathbf{x}, \hat{\boldsymbol{\omega}}_{\boldsymbol{t}}) \\ \log p(y = c|\mathbf{x}, \hat{\boldsymbol{\omega}}_{\boldsymbol{t}}) \end{aligned} \tag{2.4}$$

The difference between Equation 2.4 and 2.3 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.

## 2.4 Heuristics

### 2.4.1 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 reevaluated less frequently. This in turn results in the model getting 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, because it is uncertain about a specific type of data at that 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 would not be desirable 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.

The above could form a problem as although a smaller $q$ should theoretically provide us with better results, it also results in the uncertainties having to be computed more often. Every time the uncertainties are computed, $T$ stochastic forward passes have to be made 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 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 are recomputed. This process gets repeated until the query-pool 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 A.3 of the Appendix shows how RET is integrated in the AL loop.

RET its main purpose is enabling 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 the query-pool result in less influence of the heuristic. When these both grow in size, the computation and waiting time of the oracle in between active learning rounds rises exponentially. This is the case because of the fact that $T$*$q$ forward passes have to be made multiple times for every additional example in $RP$. 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 2.2.3). 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 the query-pool if their cosine similarity to all other points that are already in the query-pool is lower than the chosen threshold $l$. If not enough examples are selected to get the desired query-pool size, the threshold gets decreased by 0.01. Algorithm A.4 of the Appendix shows how this heuristic is added to the AL loop.

### 2.4.2 Sampling by Uncertainty and Density (SUD)

Oosten and Schomaker (2014) 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 (Roy & McCallum, 2001; Tang, Luo, & Roukos, 2002).

Outliers contain a lot of information that the model has not encountered yet, but this information is not necessarily useful. We hypothesise that, like with RECT described 2.4.1, semantically similar sentences provide the same type information. In that situation, outliers are very far from other examples in embedding space.

Zhu, Wang, Yao, and Tsou (2008) 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 datapoints is computed. If this is a low value, it indicates that the datapoint is not very similar to other values. 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 ME classifier. We will explore whether this approach also works for BERT combined with the embeddings computed by SentenceBERT. The adjusted pseudocode is shown in the Appendix (Algorithm A.5).

## 2.5 Experimental Setup[†]

### 2.5.1 Data

Two datasets were used to validate and compare the performance of the different AL implementations. Table 2.1 shows an overview of the amount of examples and classes of each dataset.

Table 2.1: An overview of the two datasets used in the experiments

| Dataset | Examples | Features |
|---|---|---|
| SST | 11,850 | 5 |
| KvK | 2212 | 15 |

The first of the used datasets was the Stanford Sentiment Treebank (Socher et al., 2013) (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:

- $0 \leq \text{label} < 0.2$: very negative
- $0.2 \leq \text{label} < 0.4$: negative
- $0.4 \leq \text{label} \leq 0.6$: neutral
- $0.6 < \text{label} \leq 0.8$: positive
- $0.8 < \text{label} \leq 1$: 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.

### 2.5.2 Evaluation Metrics

To evaluate and compare the performance of the different AL strategies, three evaluation metrics were reported: the accuracy, the ROCAUC score (Bradley, 1997), and an altered version of the deficiency metric proposed in Zhu et al. (2008).

As both datasets had more than two features, the ROCAUC score got computed for every pair of class-labels and then the weighted average of these scores was taken.

The variant of deficiency that was used is shown in Equation 2.5, 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 Zhu et al. (2008), we use the maximal achieved accuracy. This accounts for the fact that the last achieved accuracy in a classification task is not necessarily the best while

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

still returning a metric which provides a summary of the entire learning curve. A value of <1 indicates a better performance than the REF strategy wheras a value of >1 indicates a worse performance.

$$DEF(AL, R) = \frac{\sum_{t=1}^{n}(max(acc(R)) - acc_t(C)}{\sum_{t=1}^{n}(max(acc(R)) - acc_t(R))} \tag{2.5}$$

### 2.5.3 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 random sampling?

2. What is the relation between query-pool size $q$ and the achieved performance?

3. Do the proposed heuristics yield improved performance when compared to a basic uncertainty-based AL strategy?

The statistical setup used for the experiments can be found in Table 2.2. The setup for SST was based on the proposed setup in Socher et al. (2013).

Table 2.2: 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 | 7951 | 1101 | 2210 |
| | (5%) | (67%) | (9%) | (19%) |
| KvK | 111 | 1659 | 221 | 1659 |
| | (5%) | (75%) | (10%) | (10%) |

To reiterate, the following AL strategies were implemented:

1. Variation Ratio (Section 2.3.1)

2. Predictive Entropy (Section 2.3.2)

3. BALD (Section 2.3.3)

4. RET (Section 2.4.1)

5. RECS (Section 2.4.1)

6. SUD (Section 2.4.2)

To answer subquestion 1, these strategies were compared to the performance of random sampling using a $q$ of 1% of the dataset. 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. To make the results more generalisable, 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 $\tau$ 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.

### 2.5.4 Hyperparameters

An overview of all hyperparameters can be found in 2.4. Both dropout rate and $l$ were chosen based on a grid search across both datasets. The amount of stochastic forward passes $T$ was based on Ein-Dor et al. (2020) and was set to 10 across all experiments. Larger values up to 100 were tested, but lead to much larger training times and little improvement on the average accuracy or the average ROCAUC score and were thus deemed to lead to a respectively larger computational cost, especially for larger datasets, without a corresponding gain in performance.

Early stopping was applied on each training phase of the AL loop. The amount of epochs used for each dataset can be found in Table 2.3. The model that got the lowest validation loss across all epochs was used for evaluation and to compute the uncertainties. 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 (Kingma & Ba, 2015) was used for optimization and its learning rate was

Table 2.3: The amount of epochs used for early stopping for the different datasets.

| Dataset | # Epochs |
|---|---|
| SST | 15 |
| KvK | 25 |

tuned based on the CLR method (Smith, 2015). The batch size was based on the fact of it being the best performing computationally feasible batch size out of the tried batch sizes (32, 64, 128, 256). The betas and $\epsilon$ were set to their default values.

The size of $\mathcal{RP}$ was chosen arbitrarily. We leave the optimization of this parameter to future research.

Finally, Principle Component Analysis was tried for dimension reduction to determine whether reducing dimensions would result in better class-separability classes. For every datapoint in the full dataset, the classes of the ten most similar datapoints (based oncosine similarity) were determined. The number of same-class datapoints within these ten was then averaged across all datapoints and used to to decide what dimensionality was to be used.

Table 2.4: The hyperparameters and their corresponding values

| Parameter | Value |
|---|---|
| Dropout-rate | 0.2 |
| $T$ | 10 |
| $l$ | 0 |
| $\beta_1, \beta_2$ | 0.9, 0.999 |
| $\epsilon$ | $1 * 10^{-8}$ |
| Learning rate | $2 * 10^{-5}$ |
| Batch size | 128 |
| $\mathcal{RP}$ size | $1.5*q$ |
| embedding dim | 768 |

# 3 Results[‡]

This section will go onto visualise and describe the achieved results for all three experiments described in 2.5.3. 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.

## 3.1 Active Learning

Figure 3.1 shows how the query functions performed on the KvK dataset. All query functions

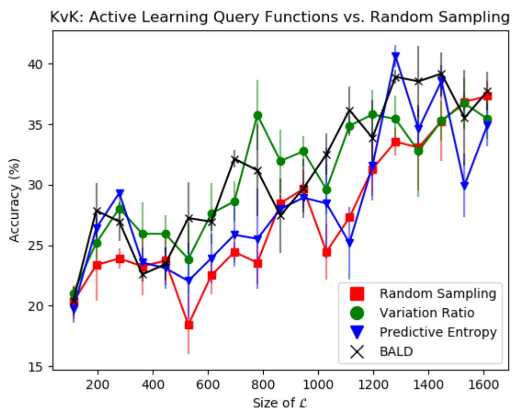


Figure 3.1: The achieved test accuracy on the KvK dataset by random sampling and the uncertainty-based query functions. The points shown are those at every 5th interval.

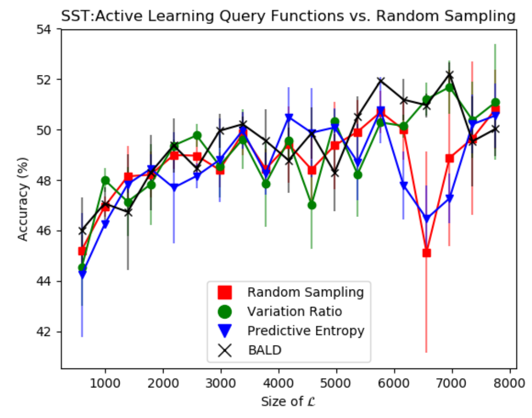
outperform random sampling when the labeled dataset is less than 200 examples large. After this, the performance of predictive entropy drops, only to reach its maximal accuracy of 40% at around 1300 labeled examples. BALD continues to outperform random sampling up until the last 200 examples while variation ratio starts performing equally to random sampling from about 1300 labeled examples. Another important insight is that variation ratio reaches close to maximal performance at about 800 labeled examples.

Figure 3.2 shows how random sampling and the implemented query functions performed on the SST dataset. When $\mathcal{L}$ reaches a size of about 3000, both predictive entropy and BALD start to outperform random sampling on ROCAUC score by a margin which falls outside of the 1 standard deviation. The difference gets largest at a size of about 6500, which is also where we see BALD and variation ratio outperform random sampling both on ROCAUC score as well as test accuracy.

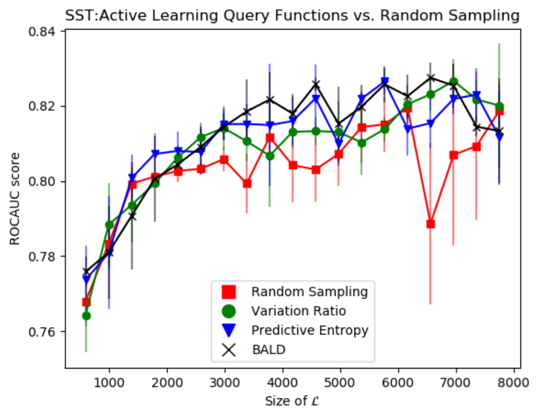
Finally, the deficiencies shown in Table 3.1 show a positive result ($< 1$) for all query functions except

[‡]No ROCAUC score was reported for the KvK dataset, as it ended up being undefined a large number of times due to only one class getting sampled in evaluation batches.

for predictive entropy for the KvK dataset. BALD has the lowest deficiency for both datasets.



(a)



(b)

Figure 3.2: The achieved test accuracy (a) and ROCAUC score (b) on the SST dataset by random sampling and the uncertainty-based query functions. The points shown are those at every 5th interval.

Table 3.1: The deficiencies 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 Scaling

Figure 3.3 shows the performance of variation ratio across different $q$ when used on the KvK dataset. In

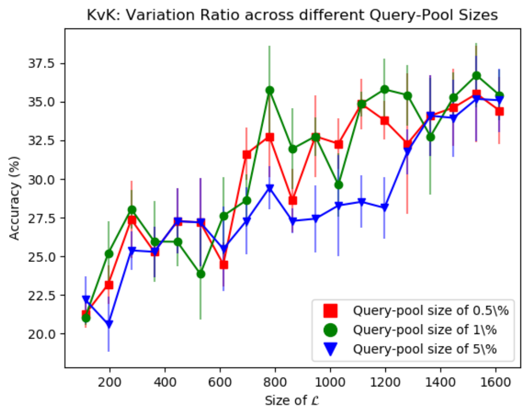


Figure 3.3: The achieved test accuracy across different query-pool sizes on the KvK dataset. The points shown of the lines representing the query pool sizes of 0.1% and 0.5% are respectively those at every 10th and 5th interval.

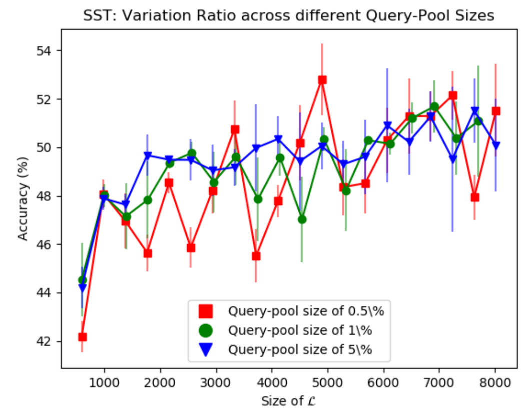
the range of 700 to 1300 examples, variation ratio with a $q$ of 5% has a worse performance than the other $q$. This difference in performance is highest at about 1200 labeled examples. The $q$ of 0.5% and 1% achieve similar performance with the accuracy scores always staying within one standard deviation of each other.

Figure 3.4 shows the performance of the different $q$ on the SST dataset. The different $q$ all achieve similar performance in terms of both accuracy and ROCAUC score. Only the $q$ of 0.5% manages to outperform the other $q$ in terms of accuracy at about 5000 labeled examples.

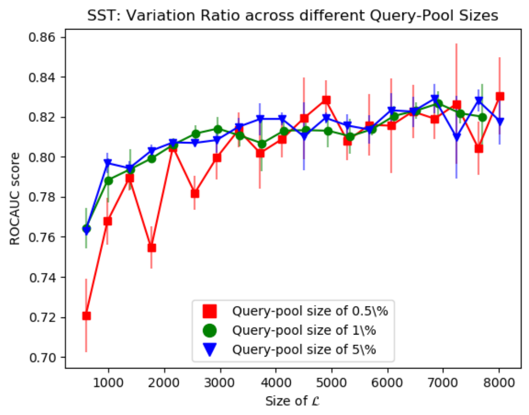
The deficiencies for the different $q$ across both datasets are shown in Table 3.2. For the SST dataset, the $q$ of 5% had a lower deficiency across the learning curve whereas the $q$ of 0.5% shows a relatively high deficiency. For the KvK dataset however, we see that the $q$ of 5% has a relatively high deficiency when compared to the similarly performing $q$ of 0.5% and 1%.

Table 3.2: The achieved deficiencies 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 |



(a)



(b)

Figure 3.4: The achieved test accuracy (a) and ROCAUC score (b) on the SST dataset by using the variation ratio query function with different $q$. The points shown of the lines representing the query pool sizes of and 0.5% and 1% are respectively those at every 10th and 5th and interval.

## 3.3 Heuristics

Figure 3.6 shows the performance of using variation ratio with heuristics together with the performance of solely using variation ratio on the KvK dataset (this is the same run shown in Figure 3.2.

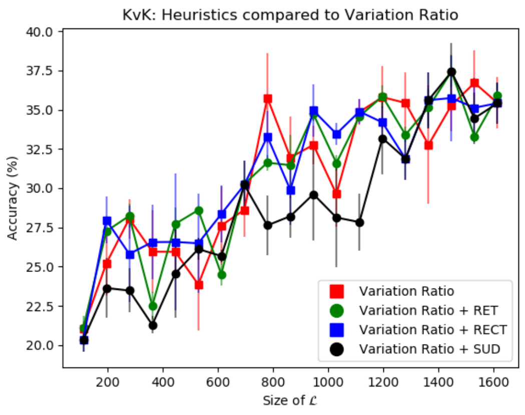


Figure 3.5: The achieved test accuracy on the KvK dataset by the different heuristics. The points shown are those at every 5th interval

Both RET and RECT show no significant improvement over solely using variation ratio. The same can be gathered from the results of the SST dataset shown in Figure 3.6, were their accuracy scores stay within one standard deviation for the entire learning curve.

Moreover, Table 3.3 shows that the average Kendall's $\tau$ is around 0 with a relatively large standard deviation. This indicates that there is no relationship between the compared rankings.

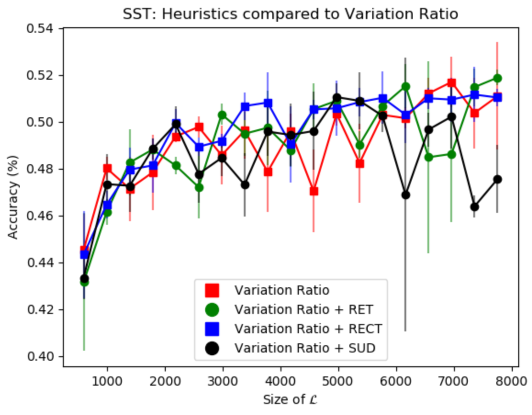
Table 3.3: The mean and the 1 standard deviation range of Kendall's $\tau$ from the described ranking experiment across the two datasets. Both measures were rounded to two decimal places.

| Dataset | Mean | $\sigma$ |
|---|---|---|
| SST | 0.14 | 0.33 |
| KvK | 0.02 | 0.47 |

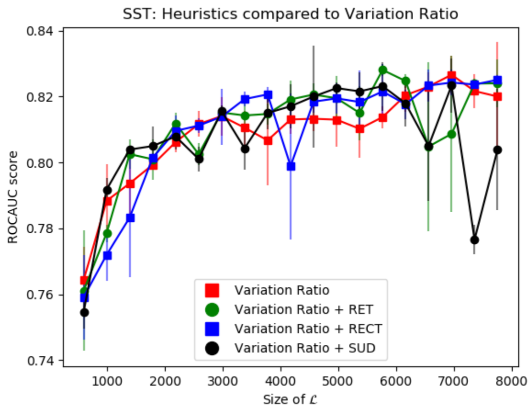
Lastly, SUD shows a worse performance for both the SST and KvK datasets. For the SST dataset, this decline in performance is seen at around 6000 labeled examples as well as from 7500 labeled examples and onwards. For the KvK dataset, it occurs between 200 and 400 labeled examples as well as between 800 and 1100 examples. The deficiencies shown in Table 3.4 also show high values for SUD across both datasets.

Table 3.4: The achieved deficiencies by the different heuristics for the different datasets. Variation ratio was the reference strategy.

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



(a)



(b)

Figure 3.6: The achieved test accuracy and ROCAUC score on the SST dataset by the different heuristics. The points shown are those at every 5th interval.

## 4 Discussion

The goal of this research was to find out whether Active Learning (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.

### 4.1 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 though: there were specific points at which AL outperformed random sampling and others at which 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.

From the above, we conclude that uncertainty-based AL with BERT$_{base}$ can be used to decrease labeling effort. This corresponds to what was concluded by Grießhaber, Maucher, and Vu (2020). We also conclude that its scalability is limited, as there can be an inverse relationship between query-pool size and performance.

When looking at the bigger picture, we showed that AL can still provide an improvement in performance over random sampling for large datasets. Unfortunately, the scaling problem could prove to be detrimental to AL being applicable to real world problems. This being due to the fact that labeling with small query-pool sizes will take a tremendous amount of time and computation for larger datasets. One might have to weigh the cost of performance against the cost of computation and longer waiting times for the oracle in between the moments he gets to label data. Additionally, the improvement of performance of AL with BERT is limited when compared to what it achieved for older NLP models (Roy & McCallum, 2001; Tang et al., 2002; Zhu et al., 2008) and even more so when compared to image classifiers (Drost, 2020; Gal et al., 2017; Houlsby et al., 2011). 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 (Gupta, Thadani, & O'Hare, 2020) 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%.

## 4.2 Limitations

Regrettably, there were some flaws in the methodology which could have had an impact on the generalizability and the reliability of the results. One of these being that, due to computational constaints, only the first sentence of texts was used. There were datapoints where the first sentence did not contain any clear indication of its label. Take for example the following description from the KvK dataset:

*"**Hi, I'm Barbara Goudsmit.** Welcome to my woven world! I am a passionate hand weaver from the Netherlands who loves creating patterns and bringing them to live on my 8-shaft loom."*

This type of data could have resulted in the network learning suboptimal mappings, which could in turn have had an influence on the performance of AL. Another limitation could be the fact that SentenceBERT was not pretrained on the task-specific labels before assigning embeddings to them. It was found that retraining SentenceBERT on the labeled dataset $\mathcal{L}$ did not improve the amount of same-class data being the closest neighbors. This, however, is just an indication of the fact that training SentenceBERT would not improve the embeddings. It could still be the case that similar examples would get closer in embedding space in a way that could improve the performance of RECT and SUD (Sections 2.4.1 and 2.4.2).

Lastly, only one seed size (5% of the dataset size) was tried across all experiments. One could argue that this does not show the robustness of the tested AL approaches and that smaller seed sizes should have been tried in the spirit of getting AL to decide what data should be used from an earlier point.

## 4.3 Future Research

This work focused on classification tasks. A future direction could be to investigate the influence of AL on BERT its performance in the context of regression tasks and to also examine how the proposed heuristics perform in that type of task.

Moreover, instead of using BERT$_{base}$, more recent BERT variants, like for instance RoBERTa (Liu et al., 2019), 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. It could prove fruitful to research query functions aimed at text classification or at the fact that BERT is a pretrained model.

Another important problem for reducing labeling effort in a real-world setting would be to research ways of deciding when the labeling process is to be ended. In this study, we ran the AL algorithms until the full dataset was labeled, but in a non-research setting one would want to know when a good approximation of the full dataset is reached to then be able to quit labeling data.

Lastly, an important direction for future work could be to try and make AL more scalable. This could be done through making larger query-pool sizes more viable. One suggestion for this would be to try and improve the proposed heuristics by trying different ways of creating the sentence embeddings.

We hope that the proposed algorithms, results, and conclusions of this paper provide a good starting point for the mentioned future studies.

## Acknowledgments

We would like to express our thanks and gratitude to prof. dr. L.R.B. Schomaker (University of Groningen, Groningen) for his helpful suggestions which ended up positively influencing the research. Also many thanks to the people at Dialogic (Utrecht) of which Nick Jelicic in particular, for the useful advice on the writing style of the paper and the suggested improvements for the source code.

## References

Ahmed, W., Natarajan, T., & Rao, K. R. (1974). Discrete Cosine Transform. *IEEE Transactions on Computers*, *23*(1), 90–93.

Bouneffouf, D., Laroche, R., Urvoy, T., Féraud, R., & Allesiardo, R. (2014, October). Contextual Bandit for Active Learning: Active Thompson Sampling. In *Proceedings of the 21st International Conference on Neural Information Processing (ICONIP)* (pp. 405–412). Springer International Publishing.

Bradley, A. P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, *30*(7), 1145–1159.

Cai, W., Zhang, Y., & Zhou, J. (2013, December). Maximizing Expected Model Change for Active Learning in Regression. In *Proceedings - IEEE International Conference on Data Mining, ICDM* (pp. 51–60).

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019, June). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *North American Association for Computational Linguistics (NAACL)* (pp. 4171–4186). Association for Computational Linguistics.

Drost, F. (2020). *Uncertainty Estimation in Deep Neural Networks for Image Classification* (Unpublished master's thesis). University of Groningen.

Ein-Dor, L., Halfon, A., Gera, A., Shnarch, E., Dankin, L., Choshen, L., … Slonim, N. (2020, November). Active Learning for BERT: An Empirical Study. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 7949–7962). Association for Computational Linguistics.

Gal, Y. (2016). *Uncertainty in Deep Learning* (Unpublished master's thesis). University of Cambridge.

Gal, Y., & Ghahramani, Z. (2016). 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.

Gal, Y., Islam, R., & Ghahramani, Z. (2017, Augustus). Deep Bayesian Active Learning with Image Data. In *International Conference on Machine Learning* (Vol. 70, pp. 1183–1192). PMLR.

Grießhaber, D., Maucher, J., & Vu, N. T. (2020). Fine-tuning BERT for Low-Resource Natural Language Understanding via Active Learning. *CoRR*, *abs/2012.02462*.

Gulati, P., Sharma, A., & Gupta, M. (2016). Theoretical Study of Decision Tree Algorithms to Identify Pivotal Factors for Performance Improvement: A Review. *International Journal of Computer Applications*, *141*(14), 19–25.

Gupta, A., Thadani, K., & O'Hare, N. (2020, December). Effective Few-Shot Classification with Transfer Learning. In *Proceedings of the 28th International Conference on Computational Linguistics* (pp. 1061–1066). International Committee on Computational Linguistics.

Houlsby, N., Huszár, F., Ghahramani, Z., & Lengyel, M. (2011). *Bayesian Active Learning for Classification and Preference Learning.*

Hu, P., Lipton, Z. C., Anandkumar, A., & Ramanan, D. (2018). Active Learning with Partial Feedback. *CoRR*, *abs/1802.07427*.

Kingma, D., & Ba, J. (2015). Adam: A Method for Stochastic Optimization. *CoRR*, *abs/1412.6980*.

Krogh, A., & Vedelsby, J. (1994). 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.

Kullback, S., & Leibler, R. A. (1951). On Information and Sufficiency. *The Annals of Mathematical Statistics*, *22*(1), 79–86.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., … Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR*, *abs/1907.11692*.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013, January). Efficient Estimation of Word Representations in Vector Space. In *Proceedings of Workshop at ICLR* (pp. 1–12).

Oosten, J.-P., & Schomaker, L. (2014). Separability versus prototypicality in handwritten word-image retrieval. *Pattern Recognition*, *47*(3), 1031–1038.

Pennington, J., Socher, R., & Manning, C. (2014, October). GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532–1543). Association for Computational Linguistics.

Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *CoRR*, *abs/1908.10084*.

Roy, N., & McCallum, A. (2001, August). Toward Optimal Active Learning through Sampling Estimation of Error Reduction. In *Proceedings of the Eighteenth International Conference on Machine Learning* (p. 441-–448). Morgan Kaufmann Publishers Inc.

Settles, B. (2012). Active Learning Literature Survey. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *6*(1), 1–114.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell System Technical Journal*, *27*(3), 379–423.

Smith, L. N. (2015). No More Pesky Learning Rate Guessing Games. *CoRR*, *abs/1506.01186*.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., & Potts, C. (2013, October). 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). Association for Computational Linguistics.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, *15*(56), 1929–1958.

Tang, M., Luo, X., & Roukos, S. (2002, July). Active Learning for Statistical Natural Language Parsing. In *Proceedings of ACL 2002* (pp. 120–127). Association for Computational Linguistics.

Teye, M., Azizpour, H., & Smith, K. (2018, July). Bayesian Uncertainty Estimation for Batch Normalized Deep Networks. In *Proceedings of the 35th International Conference on Machine Learning* (Vol. 80, pp. 4907–4916). PMLR.

Tsymbalov, E., Panov, M., & Shapeev, A. (2018). Dropout-Based Active Learning for Regression. *Analysis of Images, Social Networks and Texts*, 247-258.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., . . . Polosukhin, I. (2017). Attention is All you Need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Vol. 30, p. 6000—6010). Curran Associates, Inc.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., . . . Dean, J. (2016). Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation. *CoRR*, *abs/1609.08144*.

Zhu, J., Wang, H., Yao, T., & Tsou, B. K. (2008, August). 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). Coling 2008 Organizing Committee.

## A Algorithms

**Algorithm A.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 A.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 A.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 A.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 $Ulength < q$ 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 A.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
```