Small-Text: Active Learning for Text Classification in Python

Christopher Schröder¹, Lydia Müller¹,², Andreas Niekler¹, Martin Potthast¹,³

¹Leipzig University
²Institute for Applied Informatics (InfAI), Leipzig
³ScaDS.AI

Abstract

We introduce small-text, an easy-to-use active learning library, which offers pool-based active learning for single- and multi-label text classification in Python. It features numerous pre-implemented state-of-the-art query strategies, including some that leverage the GPU. Standardized interfaces allow the combination of a variety of classifiers, query strategies, and stopping criteria, facilitating a quick mix and match, and enabling a rapid and convenient development of both active learning experiments and applications. With the objective of making various classifiers and query strategies accessible for active learning, small-text integrates several well-known machine learning libraries, namely scikit-learn, PyTorch, and Hugging Face transformers. The latter integrations are optionally installable extensions, so GPUs can be used but are not required. Using this new library, we investigate the performance of the recently published SetFit training paradigm, which we compare to vanilla transformer fine-tuning, finding that it matches the latter in classification accuracy while outperforming it in area under the curve. The library is available under the MIT License at https://github.com/webis-de/small-text, in version 1.3.0 at the time of writing.

1 Introduction

Text classification, like most modern machine learning applications, requires large amounts of training data to achieve state-of-the-art effectiveness. However, in many real-world use cases, labeled data does not exist and is expensive to obtain, especially when domain expertise is required. Active Learning (Lewis and Gale, 1994) solves this problem by repeatedly selecting unlabeled data instances that are deemed informative according to a so-called query strategy, and then having them labeled by an expert (see Figure 1a). A new model is then trained on all previously labeled data, and this process is repeated until a specified stopping criterion is met.

Active learning aims to minimize the amount of labeled data required while maximizing the effectiveness (increase per iteration) of the model, e.g., in terms of classification accuracy.

An active learning setup, as shown in Figure 1b, generally consists of up to three components on the system side: a classifier, a query strategy, and an optional stopping criterion. Meanwhile, many approaches for each of these components have been proposed and studied. Determining appropriate combinations of these approaches is only possible experimentally, and efficient implementations are often nontrivial. In addition, the components often depend on each other, for example, when a query strategy relies on parts specific to certain model classes, such as gradients (Ash et al., 2020) or embeddings (Margatina et al., 2021). The more such non-trivial combinations are used together, the more the reproduction effort increases, making a modular library essential.

An obvious solution to the above problems is the use of open source libraries, which, among other benefits, accelerate research and facilitate technology transfer between researchers as well as into practice (Sonnenburg et al., 2007). While solu-
Figure 2: Module architecture of small-text. The core module can optionally be extended with a PyTorch and transformers integration, which enable to use GPU-based models and state-of-the-art transformer-based text classifiers of the Hugging Face transformers library, respectively. The dependencies between the module’s packages have been omitted.

Figure 2: Module architecture of small-text. The core module can optionally be extended with a PyTorch and transformers integration, which enable to use GPU-based models and state-of-the-art transformer-based text classifiers of the Hugging Face transformers library, respectively. The dependencies between the module’s packages have been omitted.

The library provides relevant text classification baselines such as SVM (Joachims, 1998) and Kim-CNN (Kim, 2014), and many more can be used through scikit-learn. Recent transformer models such as BERT (Devlin et al., 2019) are available through the transformers integration. This integration also includes a wrapper that enables the use of the recently published SetFit training paradigm (Tunstall et al., 2022), which uses contrastive learning to fine-tune SBERT embeddings (Reimers and Gurevych, 2019) in a sample efficient manner.

As the query strategy, which selects the instances to be labeled, is the most salient component of an active learning setup, the range of alternative query strategies provided covers four paradigms at the time of writing: (i) confidence-based strategies: least confidence (Lewis and Gale, 1994; Culotta and McCallum, 2005), prediction entropy (Roy and McCallum, 2001), breaking ties (Luo et al., 2005), BALD (Houlsby et al., 2011), CVIRS (Reyes et al., 2018), and contrastive active learning (Margatina et al., 2021); (ii) embedding-based strategies: BADGE (Ash et al., 2020), BERT k-means (Yuan et al., 2020), discriminative active learning (Gissin and Shalev-Shwartz, 2019), and SEALS (Coleman et al., 2022); (iii) gradient-based strategies: expected gradient length (EGL; Settles et al., 2007), EGL-word (Zhang et al., 2017), and EGL-sm (Zhang et al., 2017); and (iv) coreset strategies: greedy coreset (Sener and Savarese, 2018) and lightweight coreset (Bachem et al., 2018). Since there is an abundance of query strategies, this list will likely never be exhaustive—also because strategies from other domains, such as computer vision, are not always applicable to the text domain, e.g., when using the geometry of images (Konyushkova et al., 2015), and thus will be disregarded here.

Furthermore, small-text includes a considerable amount of different stopping criteria: (i) stabilizing predictions (Bloodgood and Vijay-Shanker, 2009), (iv) overall-uncertainty (Zhu et al., 2008), (iii) classification-change (Zhu et al., 2008),
(ii) predicted change of F-measure (Altschuler and Bloodgood, 2019), and (v) a criterion that stops after a fixed number of iterations. Stopping criteria are often neglected in active learning although they exert a strong influence on labeling efficiency.

The library is available via the python packaging index and can be installed with just a single command: pip install small-text. Similarly, the integrations can be enabled using the extra requirements argument of Python’s setuptools, e.g., the transformers integration is installed using pip install small-text[transformers]. The robustness of the implementation rests on extensive unit and integration tests. Detailed examples, an API documentation, and common usage patterns are available in the online documentation.1

3 Library versus Annotation Tool

We designed small-text for two types of settings: (i) experiments, which usually consist of either automated active learning evaluations or short-lived setups with one or more human annotators, and (ii) real-world applications, in which the final model is subsequently applied on unlabeled or unseen data. Both cases benefit from a library which offers a wide range of well-tested functionality.

To clarify on the distinction between a library and an annotation tool, small-text is a library, by which we mean a reusable set of functions and classes that can be used and combined within more complex programs. In contrast, annotation tools provide a graphical user interface and focus on the interaction between the user and the system. Obviously, small-text is still intended to be used by annotation tools but remains a standalone library. In this way it can be used (i) in combination with an annotation tool, (ii) within an experiment setting, or (iii) as part of a backend application, e.g., a web API. As a library it remains compatible to all of these use cases. This flexibility is supported by the library’s modular architecture which is also in concordance with software engineering best practices, where high cohesion and low coupling (Myers, 1975) are known to contribute towards highly reusable software (Müller et al., 1993; Tonella, 2001). As a result, small-text should be compatible with most annotations tools that are extensible and support text classification.

4 Code Example

In this section we show a code example to perform active learning with transformers models.

**Dataset** First, we create (for the sake of a simple example) a synthetic two-class spam dataset of 100 instances. The data is given by a list of texts and a list of integer labels. To define the tokenization strategy, we provide a transformers tokenizer. From these individual parts we construct a TransformersDataset object which is a dataset abstraction that can be used by the interfaces in small-text. This yields a binary text classification dataset containing 50 examples of the positive class (spam) and the negative class (ham) each:

```python
import numpy as np
from small_text import TransformersDataset, TransformerModelArguments
from transformers import ...
train = TransformersDataset.from_arrays(
text, labels, tokenizer,
target_labels=np.array([0, 1]),
max_length=10
)
```

**Active Learning Configuration** Next, we configure the classifier and query strategy. Although the active learner, query strategies, and stopping criteria components are dataset- and classifier-agnostic, classifier and dataset have to match (i.e. TransformerBasedClassification must be used with TransformersDataset) owing to the different underlying data structures:

```python
from small_text import LeastConfidence, TransformerBasedClassificationFactory as TransformerFactory
clf_factory = TransformerFactory(model_args, num_classes, kwargs={'device': 'cuda'})
query_strategy = LeastConfidence()
```

---

1https://small-text.readthedocs.io
Since the active learner may need to instantiate a new classifier before the training step, a factory\cite{gamma1995design} is responsible for creating new classifiers. Finally, we set the query strategy to least confidence\cite{culotta2005active}.

**Initialization** There is a chicken-and-egg problem for active learning because most query strategies rely on the model, and a model in turn is trained on labeled instances which are selected by the query strategy. This problem can be solved by either providing an initial model\eg through manual labeling, or by using cold start approaches\cite{yuan2020active}. In this example we simulate a user-provided initialization by looking up the respective true labels and providing an initial model:

```python
from small_text import \nPoolBasedActiveLearner, \nrandom_initialization_balanced as init
active_learner = ... = init(train.y, n_samples=10)
active_learner.initialize_data(
indices_initial,
train.y[indices_initial]
)
```

To provide an initial model in the experimental scenario\where true labels are accessible\smalltext provides sampling methods, from which we use the balanced sampling to obtain a subset whose class distribution is balanced\or close thereto. In a real-world application, initialization would be accomplished through a starting set of labels supplied by the user. Alternatively, a cold start classifier or query strategy can be used instead.

**Active Learning Loop** After the previous code examples prepared the setting by loading a dataset, configuring the active learning setup, and providing an initial model, the following code block shows the actual active learning loop. In this example, we perform five queries during each of which ten instances are queried. During a query step the query strategy samples instances to be labeled. Subsequently, new labels for each instance are provided and passed to the update method, and then a new model is trained. In this example, it is a simulated response relying on true labels, but in a real-world application this part is the user’s response.

```python
from sklearn.metrics import accuracy_score
num_queries = 5
for i in range(num_queries):
    # Query 10 samples per iteration.
    indices_queried = active_learner.query(num_samples=10)
    indices_initial = init(train.y, n_samples=10)
    active_learner.initialize_data(indices_initial, train.y[indices_initial])
    y = train.y[indices_queried]
    # Provide labels for the queried indices.
    active_learner.update(y)
    # Evaluate accuracy on the train set
    y_pred = active_learner.classifier.predict(train)
    print(f'Train accuracy: {accuracy_score(y_pred, train.y)}')
print(f'Iteration {i+1}')
print('Train accuracy: {:.2f}'.format(accuracy_score(y_pred, train.y)))
```

In summary, we built a full active learning setup in only very few lines of code. The actual active learning loop consists of just the previous code block and changing hyperparameters,\eg using a different query strategy, is as easy as adapting the query_strategy variable.

## 5 Comparison to Previous Software

Unsurprisingly, after decades of research and development on active learning, numerous other libraries are available that focus on active learning as well. In the following we present a selection of the most relevant open-source projects for which either a related publication is available or a larger user base exists: JCLAL\cite{reyes2016jclal} is a generic framework for active learning which is implemented in Java and can be used either through XML configurations or directly from the code. It offers an experimental setting which includes 18 query strategies. The aim of libact\cite{yang2017libact} is to provide active learning for real-world applications. Among 19 other strategies, it includes a well-known meta-learning strategy\cite{hsu2015meta}. Baal\cite{atighehchian2020baal} provides bayesian active learning including methods to obtain uncertainty estimates. The modAL library\cite{danka2018modal} offers single- and multi-label active learning, provides 21 query strategies, also builds on scikit-learn by default, and offers instructions how to include GPU-based models using Keras and PyTorch. ALiPy\cite{tang2019ali} provides an active learning framework targeted at
the experimental active learning setting. Apart from providing 22 query strategies, it supports alternative active learning settings, e.g., active learning with noisy annotators. The low-resource-text-classification-framework (lrtc; (Ein-Dor et al., 2020)) is an experimentation framework for the low resource scenario and supports which can be easily extended. It also focuses on text classification and has a number of built-in models, datasets, and query strategies to perform active learning experiments. Another recent library is scikit-activeml which offers general active learning built around scikit-learn. It comes with 29 query strategies but provides no stopping criteria. GPU-based functionality can be used via skorch, a PyTorch wrapper, which is a ready-to-use adapter as opposed to our implemented classifier structures but is on the other hand restricted to the scikit-learn interfaces. ALToolbox (Tsvigun et al., 2022) is an active learning framework that provides an annotation interface and a benchmarking mechanism to develop new query strategies. While it has some overlap with small-text, it is not a library, but also focuses on text data, namely on text classification and sequence tagging.

In Table 1, we compare small-text to the previously mentioned libraries, and compare them based on several criteria related to active learning or to the respective code base: While all libraries provide a selection of query strategies, not all libraries offer stopping criteria, which are crucial to reducing the total annotation effort and thus directly influence the efficiency of the active learning process (Vlachos, 2008; Laws and Schütze, 2008; Olsson and Tomanek, 2009). We can also see a difference in the number of provided query strategies. While a higher number of query strategies is certainly not a disadvantage, it is more important to provide the most relevant strategies (either due to recency, domain-specificity, strong general performance, or because it is a baseline). Based on these criteria, small-text provides numerous recent strategies such as BADGE (Ash et al., 2020), BERT K-Means (Yuan et al., 2020), and contrastive active learning (Margatina et al., 2021), as well as the gradient-based strategies by Zhang et al. (2018). The distinguishing characteristic of small-text is the focus on text classification, paired with a multitude of interchangeable components. It of-

<table>
<thead>
<tr>
<th>Name</th>
<th>Active Learning</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>QS</td>
<td>SC</td>
</tr>
<tr>
<td>JCLAL</td>
<td>18</td>
<td>2</td>
</tr>
<tr>
<td>libact</td>
<td>19</td>
<td>-</td>
</tr>
<tr>
<td>modAL</td>
<td>21</td>
<td>-</td>
</tr>
<tr>
<td>ALiPy</td>
<td>22</td>
<td>4</td>
</tr>
<tr>
<td>BaaL</td>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>lrtc</td>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>scikit-activeml</td>
<td>29</td>
<td>-</td>
</tr>
<tr>
<td>ALToolbox</td>
<td>19</td>
<td>-</td>
</tr>
<tr>
<td>small-text</td>
<td>14</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 1: Comparison between small-text and relevant previous active learning libraries. We abbreviated the number of query strategies by “QS”, the number of stopping criteria by “SC”, and the low-resource-text-classification-framework by lrtc. All information except “Publication Year” and “Code Repository” has been extracted from the linked GitHub repository of the respective library on February 24th, 2023. Random baselines were not counted towards the number of query strategies. Publications: 1Reyes et al. (2016), 2Yang et al. (2017), 3Danka and Horvath (2018), 4Tang et al. (2019), 5Atighehchian et al. (2020), 6Ein-Dor et al. (2020), 7Kotike et al. (2021), 8Tsvigun et al. (2022).
fers the most comprehensive set of features (as shown in Table 1) and through the integrations these components can be mixed and matched to easily build numerous different active learning setups, with or without leveraging the GPU. Finally, it allows to use concepts from natural language processing (such as transformer models) and provides query strategies unique to text classification.

6 Experiment

We perform an active learning experiment comparing an SBERT model trained with the recent sentence transformers fine-tuning paradigm (SetFit; (Tunstall et al., 2022)) over a BERT model trained with standard fine-tuning. SetFit is a contrastive learning approach that trains on pairs of (dis)similar instances. Given a fixed amount of differently labeled instances, the number of possible pairs is considerably higher than the size of the original set, making this approach highly sample efficient (Chuang et al., 2020; Hénaff, 2020) and therefore interesting for active learning.

Setup We reproduce the setup of our previous work (Schröder et al., 2022) and evaluate on the datasets shown in Table 2 with an extended set of query strategies. Starting with a pool-based active learning setup with 25 initial samples, we perform 20 queries during each of which 25 instances are queried and labeled. Since SetFit has only been evaluated for single-label classification (Tunstall et al., 2022), we focus on single-label classification as well. The goal is to compare the following two models: (i) BERT (bert-large-uncased; (Devlin et al., 2019)) with 336M parameters and (ii) SBERT (paraphrase-mpnet-base-v2; (Reimers and Gurevych, 2019)) with 110M parameters. The first model is trained via vanilla fine-tuning and the second using SetFit. For the sake of brevity, we refer to the first as “BERT” and to the second as “SetFit”. To compare their performance during active learning, we provide an extensive benchmark over multiple computationally inexpensive uncertainty-based query strategies, which were selected due to encouraging results in our previous work. Moreover, we include BALD, BADGE, and greedy coreset—all of which are computationally more expensive, but have been increasingly used in recent work (Ein-Dor et al., 2020; Yu et al., 2022).

Results In Table 3, the results show the summarized classification performance in terms of (i) final accuracy after the last iteration, and (ii) area under curve (AUC). We also compare strategies by ranking them from 1 (best) to 8 (worst) per model and dataset by accuracy and AUC. First, we can also confirm for SetFit the earlier finding that uncertainty-based strategies perform strong for BERT (Schröder et al., 2022). Second, SetFit configurations result in between 0.06 and 1.7 percentage points higher mean accuracy, and also in between 4.2 and 6.6 higher AUC when averaged over model and query strategy. Interestingly, the greedy coreset strategy (CS) is remarkably more successful for the SetFit runs compared to the BERT runs. Detailed results per configuration can be found

\[ \text{Table 2: Key characteristics about the examined datasets:}  \]

- AG’s News\(^1\) (AGN)
- Customer Reviews\(^2\) (CR)
- Movie Reviews\(^3\) (MR)
- Subjectivity\(^4\) (SUBJ)
- TREC-6\(^5\) (TREC-6)

\[ \text{Table 3: The “Rank” columns show the mean rank when ordered by mean accuracy (Acc.) and by area under curve (AUC). The “Result” columns show the mean accuracy and AUC. All values used in this table refer to state after the final iteration. Query strategies are abbreviated as follows: prediction entropy (PE), breaking ties (BT), least confidence (LC), contrastive active learning (CA), BALD (BA), BADGE (BD), greedy coreset (CS), and random sampling (RS).} \]

---

### Table 2: Key characteristics about the examined datasets:

<table>
<thead>
<tr>
<th>Dataset Name</th>
<th>Type</th>
<th>Classes</th>
<th>Training</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>AG’s News(^1) (AGN)</td>
<td>N</td>
<td>4</td>
<td>120,000 *7,600</td>
<td></td>
</tr>
<tr>
<td>Customer Reviews(^2) (CR)</td>
<td>S</td>
<td>2</td>
<td>3,397 378</td>
<td></td>
</tr>
<tr>
<td>Movie Reviews(^3) (MR)</td>
<td>S</td>
<td>2</td>
<td>9,596 1,066</td>
<td></td>
</tr>
<tr>
<td>Subjectivity(^4) (SUBJ)</td>
<td>S</td>
<td>2</td>
<td>9,000 1,000</td>
<td></td>
</tr>
<tr>
<td>TREC-6(^5) (TREC-6)</td>
<td>Q</td>
<td>6</td>
<td>5,500 *500</td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{Table 3: The “Rank” columns show the mean rank when ordered by mean accuracy (Acc.) and by area under curve (AUC). The “Result” columns show the mean accuracy and AUC. All values used in this table refer to state after the final iteration. Query strategies are abbreviated as follows: prediction entropy (PE), breaking ties (BT), least confidence (LC), contrastive active learning (CA), BALD (BA), BADGE (BD), greedy coreset (CS), and random sampling (RS).} \]

---

### Table 3: The “Rank” columns show the mean rank when ordered by mean accuracy (Acc.) and by area under curve (AUC). The “Result” columns show the mean accuracy and AUC. All values used in this table refer to state after the final iteration. Query strategies are abbreviated as follows: prediction entropy (PE), breaking ties (BT), least confidence (LC), contrastive active learning (CA), BALD (BA), BADGE (BD), greedy coreset (CS), and random sampling (RS).
in the appendix, where it can be seen that SetFit reaches higher accuracy scores in most configurations, and better AUC scores in all configurations.

Discussion When trained with the new SetFit paradigm, models having only a third of the parameters compared to the large BERT model achieve results that are not only competitive, but slightly better regarding final accuracy and considerably better in terms of AUC. Since the final accuracy values are often within one percentage point or less to each other, it is obvious that the improvement in AUC stems from improvements in earlier queries, i.e. steeper learning curves. We suspect that this is at least partly owed to sample efficiency from SetFit’s training that uses pairs of instances. Moreover, this has the additional benefit of reducing instability of transformer models (Mosbach et al., 2021) as can be exemplarily seen in Figure 3. This increasingly occurs when the training set is small (Mosbach et al., 2021), which is likely alleviated with the additional instance pairs. On the other hand, training cost increase linearly with the number of pairs per instance. In the low-data regime, however, this is a manageable additional cost that is worth the benefits.

7 Library Adoption

As recent publications have already adopted small-text, we present four examples which have already successfully utilized it for their experiments.

Abusive Language Detection Kirk et al. (2022) investigated the detection of abusive language using transformer-based active learning on six datasets of which two exhibited a balanced and four an imbalanced class distribution. They evaluated a pool-based binary active learning setup, and their main finding is that, when using active learning, a model for abusive language detection can be efficiently trained using only a fraction of the data.

Classification of Citizens’ Contributions In order to support the automated classification of German texts from online citizen participation processes, Romberg and Escher (2022) used active learning to classify texts collected by three cities into eight different topics. They evaluated this real-world dataset both as a single- and multi-label active learning setup, finding that active learning can considerably reduce the annotation efforts.

Softmax Confidence Estimates Gonsior et al. (2022) examined several alternatives to the softmax function to obtain better confidence estimates for active learning. Their setup extended small-text to incorporate additional softmax alternatives and found that confidence-based methods mostly selected outliers. As a remedy to this they proposed and evaluated uncertainty clipping.

Revisiting Uncertainty-Based Strategies In a previous publication, we reevaluated traditional uncertainty-based query strategies with recent transformer models (Schröder et al., 2022). We found that uncertainty-based methods can still be highly effective and that the breaking ties strategy is a drop-in replacement for prediction entropy.

Not only have all of these works successfully applied small-text to a variety of different problems, but each work is also accompanied by a GitHub repository containing the experiment code, which is the outcome we had hoped for. We expect that small-text will continue to gain adoption within the active learning and text classification communities, so that future experiments will increasingly rely on it by both reusing existing components and by creating their own extensions, thereby supporting the field through open reproducible research.

8 Conclusion

We introduced small-text, a modular Python library, which offers state-of-the-art active learning for text classification. It integrates scikit-learn, PyTorch, and transformers, and provides robust components that can be mixed and matched to quickly apply active learning in both experiments and applications, thereby making active learning easily accessible to the Python ecosystem.
Limitations

Although a library can, among other things, lower the barrier of entry, save time, and speed up research, this can only be leveraged with basic knowledge of the Python programming language. All included algorithmic components are subject to their own limitations, e.g., the greedy coreset strategy quickly becomes computationally expensive as the amount labeled data increases. Moreover, some components have hyperparameters which require an understanding of the algorithm to achieve the best classification performance. In the end, we provide a powerful set of tools which still has to be properly used to achieve the best results.

As small-text covers numerous text classification models, query strategies, and stopping criteria, some limitations from natural language processing, text classification and active learning apply as well. For example, all included classification models rely on tokenization, which is inherently more difficult for languages which have no clear word boundaries such as Chinese, Japanese, Korean, or Thai.

Ethics Statement

In this paper, we presented small-text, a library which can—like any other software—be used for good or bad. It can be used to bootstrap classification models in scenarios where no labeled data is available. This could be used for good, e.g. for spam detection, hate speech detection, or targeted news filtering, but also for bad, e.g., for creating models that detect certain topics that are to be censored in authoritarian regimes. While such systems already exist and are of sophisticated quality, small-text is unlikely to change anything at this point. On the contrary, being open-source software, these methods can now be used by a larger audience, which contributes towards the democratization of classification algorithms.

Acknowledgments

We thank the anonymous reviewers for their constructive advice and the early adopters of the library for their invaluable feedback.

This research was partially funded by the Development Bank of Saxony (SAB) under project numbers 100335729 and 100400221. Computations were done (in part) using resources of the Leipzig University Computing Centre.

References


Supplementary Material

A Technical Environment

All experiments were conducted within a Python 3.8 environment. The system had CUDA 11.1 installed and was equipped with an NVIDIA GeForce RTX 2080 Ti (11GB VRAM).

B Experiments

Each experiment configuration represents a combination of model, dataset and query strategy, and has been run for five times.

B.1 Datasets

We used datasets that are well-known benchmarks in text classification and active learning. All datasets are accessible to the Python ecosystem via Python libraries that provide fast access to those datasets. We obtained CR and SUBJ using gluonnlp, and AGN, MR, and TREC using huggingface datasets.

B.2 Pre-Trained Models

In the experiments, we fine-tuned (i) a large BERT model (bert-large-uncased) and (ii) an SBERT paraphrase-mpnet-base model (sentence-transformers/paraphrase-mpnet-base-v2). Both are available via the huggingface model repository.

B.3 Hyperparameters

Maximum Sequence Length We set the maximum sequence length to the minimum multiple of ten, so that 95% of the given dataset’s sentences contain at most that many tokens.

Transformer Models For BERT, we adopt the hyperparameters from Schröder et al. (2022). For SetFit, we use the same learning rate and optimizer parameters but we train for only one epoch.

C Evaluation

In Table 4 and Table 5 we report final accuracy and AUC scores including standard deviations, measured after the last iteration. Note that results obtained through PE, BT, and LC are equivalent for binary datasets.

C.1 Evaluation Metrics

Active learning was evaluated using standard metrics, namely accuracy and area under the learning curve. For both metrics, the respective scikit-learn implementation was used.
As mentioned in Section 7, the experiment code of BADGE (BD), greedy coreset (CS), and random sampling (RS). The best result per dataset is printed in bold.

Table 5: Final area under curve (AUC) per dataset, model, and query strategy. We report the mean and standard deviation over five runs. The best result per dataset is printed in bold. Query strategies are abbreviated as follows: prediction entropy (PE), breaking ties (BT), least confidence (LC), contrastive active learning (CA), BALD (BA), BADGE (BD), greedy coreset (CS), and random sampling (RS). The best result per dataset is printed in bold.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>PE</th>
<th>BT</th>
<th>LC</th>
<th>CA</th>
<th>BA</th>
<th>BD</th>
<th>CS</th>
<th>RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGN</td>
<td>BERT</td>
<td>0.898</td>
<td>0.901</td>
<td>0.900</td>
<td>0.889</td>
<td>0.889</td>
<td>0.894</td>
<td>0.881</td>
<td>0.886</td>
</tr>
<tr>
<td></td>
<td>SetFit</td>
<td>0.900</td>
<td>0.902</td>
<td>0.902</td>
<td>0.892</td>
<td>0.887</td>
<td>0.896</td>
<td>0.894</td>
<td>0.877</td>
</tr>
<tr>
<td>CR</td>
<td>BERT</td>
<td>0.920</td>
<td>0.920</td>
<td>0.916</td>
<td>0.917</td>
<td>0.919</td>
<td>0.911</td>
<td>0.915</td>
<td>0.902</td>
</tr>
<tr>
<td></td>
<td>SetFit</td>
<td>0.937</td>
<td>0.937</td>
<td>0.937</td>
<td>0.938</td>
<td>0.934</td>
<td>0.913</td>
<td>0.939</td>
<td>0.912</td>
</tr>
<tr>
<td>MR</td>
<td>BERT</td>
<td>0.850</td>
<td>0.850</td>
<td>0.846</td>
<td>0.844</td>
<td>0.859</td>
<td>0.835</td>
<td>0.843</td>
<td>0.831</td>
</tr>
<tr>
<td></td>
<td>SetFit</td>
<td>0.871</td>
<td>0.871</td>
<td>0.871</td>
<td>0.869</td>
<td>0.867</td>
<td>0.864</td>
<td>0.870</td>
<td>0.871</td>
</tr>
<tr>
<td>SUBJ</td>
<td>BERT</td>
<td>0.959</td>
<td>0.959</td>
<td>0.958</td>
<td>0.958</td>
<td>0.959</td>
<td>0.948</td>
<td>0.957</td>
<td>0.937</td>
</tr>
<tr>
<td></td>
<td>SetFit</td>
<td>0.962</td>
<td>0.962</td>
<td>0.962</td>
<td>0.960</td>
<td>0.966</td>
<td>0.942</td>
<td>0.963</td>
<td>0.932</td>
</tr>
<tr>
<td>TREC-6</td>
<td>BERT</td>
<td>0.960</td>
<td>0.966</td>
<td>0.960</td>
<td>0.965</td>
<td>0.958</td>
<td>0.958</td>
<td>0.952</td>
<td>0.947</td>
</tr>
<tr>
<td></td>
<td>SetFit</td>
<td>0.966</td>
<td>0.961</td>
<td>0.966</td>
<td>0.963</td>
<td>0.961</td>
<td>0.958</td>
<td>0.967</td>
<td>0.946</td>
</tr>
</tbody>
</table>

Table 4: Final accuracy per dataset, model, and query strategy. We report the mean and standard deviation over five runs. The best result per dataset is printed in bold. Query strategies are abbreviated as follows: prediction entropy (PE), breaking ties (BT), least confidence (LC), contrastive active learning (CA), BALD (BA), BADGE (BD), greedy coreset (CS), and random sampling (RS). The best result per dataset is printed in bold.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Model</th>
<th>PE</th>
<th>BT</th>
<th>LC</th>
<th>CA</th>
<th>BA</th>
<th>BD</th>
<th>CS</th>
<th>RS</th>
</tr>
</thead>
<tbody>
<tr>
<td>AGN</td>
<td>BERT</td>
<td>0.827</td>
<td>0.839</td>
<td>0.836</td>
<td>0.821</td>
<td>0.819</td>
<td>0.840</td>
<td>0.804</td>
<td>0.825</td>
</tr>
<tr>
<td></td>
<td>SetFit</td>
<td>0.881</td>
<td>0.889</td>
<td>0.885</td>
<td>0.879</td>
<td>0.869</td>
<td>0.881</td>
<td>0.881</td>
<td>0.867</td>
</tr>
<tr>
<td>CR</td>
<td>BERT</td>
<td>0.885</td>
<td>0.885</td>
<td>0.881</td>
<td>0.881</td>
<td>0.882</td>
<td>0.876</td>
<td>0.874</td>
<td>0.877</td>
</tr>
<tr>
<td></td>
<td>SetFit</td>
<td>0.925</td>
<td>0.925</td>
<td>0.925</td>
<td>0.927</td>
<td>0.924</td>
<td>0.910</td>
<td>0.930</td>
<td>0.908</td>
</tr>
<tr>
<td>MR</td>
<td>BERT</td>
<td>0.819</td>
<td>0.819</td>
<td>0.820</td>
<td>0.813</td>
<td>0.817</td>
<td>0.808</td>
<td>0.804</td>
<td>0.813</td>
</tr>
<tr>
<td></td>
<td>SetFit</td>
<td>0.859</td>
<td>0.859</td>
<td>0.859</td>
<td>0.859</td>
<td>0.858</td>
<td>0.855</td>
<td>0.858</td>
<td>0.857</td>
</tr>
<tr>
<td>SUBJ</td>
<td>BERT</td>
<td>0.944</td>
<td>0.944</td>
<td>0.943</td>
<td>0.940</td>
<td>0.939</td>
<td>0.929</td>
<td>0.934</td>
<td>0.924</td>
</tr>
<tr>
<td></td>
<td>SetFit</td>
<td>0.953</td>
<td>0.953</td>
<td>0.953</td>
<td>0.952</td>
<td>0.950</td>
<td>0.940</td>
<td>0.949</td>
<td>0.932</td>
</tr>
<tr>
<td>TREC-6</td>
<td>BERT</td>
<td>0.818</td>
<td>0.855</td>
<td>0.837</td>
<td>0.829</td>
<td>0.816</td>
<td>0.856</td>
<td>0.799</td>
<td>0.843</td>
</tr>
<tr>
<td></td>
<td>SetFit</td>
<td>0.910</td>
<td>0.934</td>
<td>0.919</td>
<td>0.917</td>
<td>0.907</td>
<td>0.934</td>
<td>0.927</td>
<td>0.927</td>
</tr>
</tbody>
</table>

D Library Adoption

As mentioned in Section 7, the experiment code of previous works documents how small-text was used and can be found at the following locations:

- Abusive Language Detection:
  https://github.com/HannahKirk/ActiveTransformers-for-AbusiveLanguage

- Classification of Citizens’ Contributions:
  https://github.com/juliaramberg/egov-2022

- Softmax Confidence Estimates:
  https://github.com/jgonsior/btw-softmax-cli

- Revisiting Uncertainty-Based Strategies:
  https://github.com/webis-de/ACL-22