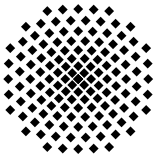

University of Stuttgart
Institute for Signal Processing and System Theory
Professor Dr.-Ing. B. Yang

Masterarbeit D1337

Learning active learning in the batch-mode setup

Lernen Activen Lernens im Batch-Modus

Author: Malte Ebner

Date of work begin: 28.11.2019
Date of submission: 10.06.2020

Supervisor: Prof. Bin Yang

Keywords: active learning, meta-learning, Markov decision process, reinforcement learning

Supervised learning models perform best when trained on a lot of data, but annotating training data is very costly in some domains. Active learning aims to select only the most informative subset of unlabelled samples for annotation, thus saving annotation cost. Several heuristics for choosing this subset have been developed, but they lack an adaptability to tasks and datasets. Thus, they have recently been replaced by functions learning the best selection from data. This is achieved by formulating active learning as a Markov decision process and applying reinforcement learning methods to it. However, recent literature on learning active learning only covers the sequential case, while batch-mode active learning has many practical advantages.
This thesis aims to fill this gap by developing and evaluating an agent learning active learning in the batch-mode setting. First, it proposes to model batch-mode active learning as a sequential batch-filling task, which makes the decision space for the agent exponentially smaller, and thus facilitates training the agent. Second, it shows that an agent learning active learning in the batch-mode setting is able to outperform heuristics consistently across different datasets and supervised learning models, which proves its adaptability.

Contents

1. Introduction

Supervised machine learning systems perform best when trained on a large amount of training data. While this training data can be obtained cheaply in some domains, labelling can cause huge time and cost efforts in other domains. Active learning in the selective scenario overcomes this bottleneck by selecting a subset of all unlabelled samples to be labelled such that the model trained on them learns as much as possible and achieves a high accuracy. This choice is done iteratively in a cycle consisting of four steps per iteration:

First the active learning agent chooses which unlabelled samples are to be labelled next. These samples are sent as a query to the oracle, e.g. a human, who labels the samples in the query. Next, the labelled samples are added to the labelled training set and removed from the set of unlabelled samples. Finally, the supervised learning algorithm is trained on the newly added samples.

This cycle is shown in Fig. 1.1. Usually the iterations end when an annotation budget is exhausted, e.g. after a certain number of samples have been labelled. An overview of active learning and a literature review is given by Settles [1].

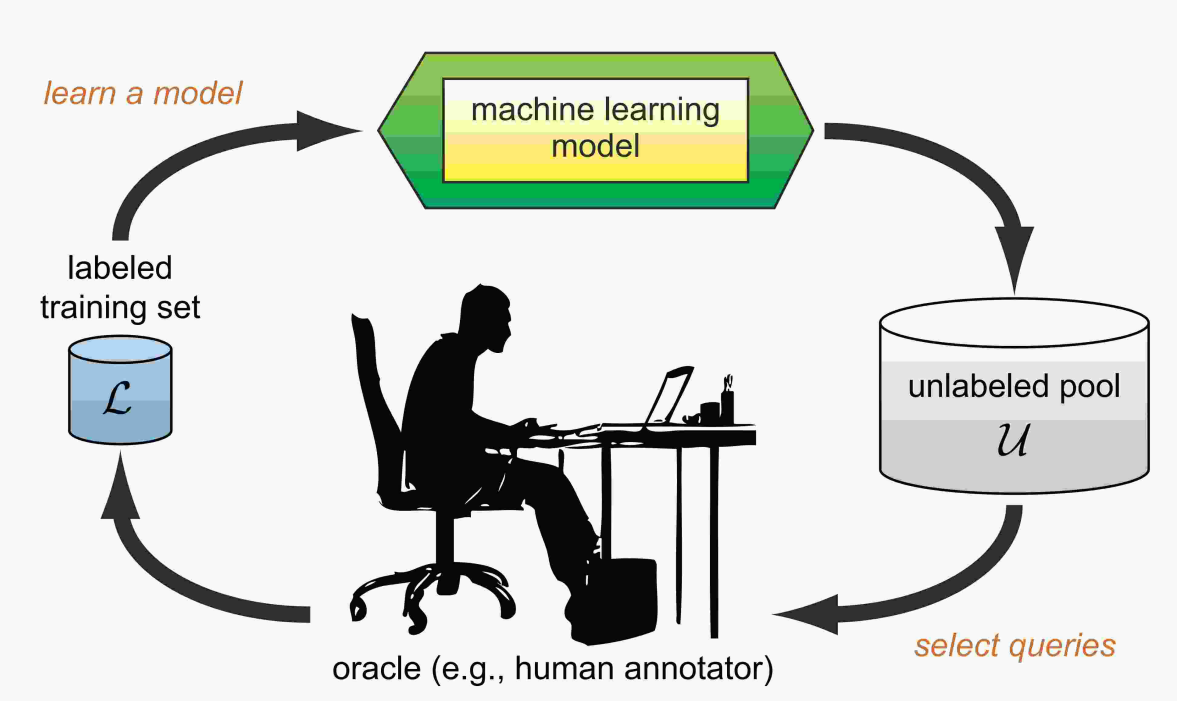

Figure 1.1.: Iteration of active learning: The cycle of selecting samples, annotating them, adding them to the labelled set and retraining the model is repeated until the desired accuracy is achieved and/or the annotation budget is exhausted.

Source: [1]

Until recently, heuristics were used as active learning agents, which chose the next sample to be labelled based on metrics like the uncertainty of the prediction or diversity metrics. [2] They have the disadvantage of using only a small subset of all available information and neither adapt to the machine learning model used, nor to the dataset, nor to the metric to be optimized. To overcome these shortcomings, state-of-the-art active leaning agents such as those in [3], [4], [5] and [6] have been developed, which try to learn the optimal policy by describing active learning as a Markov decision process (MDP) [7] and then learning the optimal policy given this MDP. In the common cold-start active learning setting, they are trained to find the optimal policy using at least one fully labelled training dataset and are then applied on a newly, unlabelled dataset. They are more adaptive to the task, the dataset and the machine learning model used than heuristics. The current relevant literature on active learning and in particular on batch-mode active learning and learning active learning is discussed in more detail in chapter 2. Learning active learning agents have only been applied to sequential active learning scenarios so far, the more practical batch-mode active learning has only been covered by heuristics like in [8], [9], [10] and [11].

This thesis aims to fill this literature gap by modelling batch-mode active learning as a Markov decision process and then training an agent on this problem. It covers the most common case of active learning: The pool-based scenario with a fixed annotation budget and constant annotation costs across all samples [2], but uses a batch-mode instead of sequential selection of samples to be labelled. Since the selection of a whole batch out of an unlabelled pool for the next annotation, i.e. choosing a subset of a large set, is a very hard and high-dimensional discrete decision problem, we propose to relax this problem to a sequential batch-filling problem: The active learning agent chooses only the next sample to add to the batch in one step, making the decision problem a single-dimensional categorical decision problem. The samples in the batch are still only annotated and the machine learning model trained on the new data when the batch is full, thus the practical advantages of batch-mode active learning are preserved. A thorough description of the MDP is given in section 3.1.

Our first approach to learning the optimal policy within the active learning MDP uses Q-Learning [12, p 157ff]: It attempts to predict the improvement of the performance of the supervised learning model when a sample is added to the batch and then chooses the sample maximizing the predicted performance improvement. The agent is trained on an active learning problem with one supervised learning model and one dataset and then applied on an active learning problem with the same supervised learning model, but a different dataset. This procedure is called 'cold-start' learning active learning, as the agent is applied to a dataset it was not trained on, thus requiring it to perform transfer learning. Several regression models were tried for fitting the Q-function, among which a random forest [13] performed best. During the training of the reinforcement learning agent several methods for on-policy and off-policy learning were tried, among which an epsilon-greedy policy performed best. The methods used are described in detail in section 3.3.

However, the Q-learning agent showed very poor reproducibility, thus we also tried a second approach: We defined a parametrised linear policy for choosing the best samples to be labelled based on features. The best parameters for it are found using Monte Carlo policy search. [12, p 113ff] This approach can be interpreted not only as a reinforcement learning approach, but also as an approach using a weighted ensemble of heuristics to choose the next sample to be labelled. The weights of the ensemble are learned on a training task. This approach performs well across different datasets and supervised learning models if it is trained

on a task similar to the one it is applied to.

Since the Monte Carlo approach has the theoretical disadvantage of choosing samples greedily, we also tried a third approach combining clustering-based non-greedy sample selection with uncertainty sampling, by learning to trade-off uncertainty measures with diversity and representative sampling. Such a combination of uncertainty sampling and clustering-based methods has already been proposed as a fruitful research direction by Sener and Savarese [14].

The agents developed in this thesis are applied to tasks from three different domains: Binary classification problems on unstructured data using a random forest [13] as classifier, classification problems on image data using a convolutional neural network [15] as classifier and a question answering problem that needs natural language processing and uses a long short term memory neural network [16] as classifier.

The experiments have shown that both the approach using Monte Carlo policy search and the one using clustering outperform heuristics consistently across this variety of domains and classifiers, sometimes by a large margin, even though they choose a large batch of samples to be labelled. It was further shown that the optimal policy varies greatly between different domains and classifiers, indicating that there is no single best active learning agent, but rather that it should be learned on a training task which agent performs best and then the optimal agent can be applied to a similar evaluation task.

2. Related Literature on Active Learning

This chapter starts with shortly outlining other methods for low-resource settings like data augmentation and semi-supervised learning, and then gives an overview of the different kind of scenarios, setups and objectives of active learning. Then it explains currently used heuristics for active learning, and their three key ideas, namely model-based, diversity and representative sampling. It compares the advantages and disadvantages of heuristics using these key ideas both theoretically and using an example classification problem to get an intuition why a combination of these three ideas is necessary to have a robustly well-performing active learning agent. Such a combination can be used by agents learning active learning, which are described in the last section of this chapter.

2.1. Other methods for low-resource settings

There are two other approaches for low-resource settings, which can be used additionally or alternatively to active learning:

Data augmentation is a technique to create additional training data out of given data. It assumes that there are variations of a training sample's input which do not change its class. According to Perez and Wang [17] there are three main approaches for data augmentation: Traditional transformations are to add noise to the input data or to shift or crop an image [18]. A second approach is to use generative adversarial networks to generate images from different styles out of the original one [19]. A third approach is to treat data augmentation as a machine learning problem and learn to augment the data. The augmented samples are used as additional labelled training samples.

Both semi-supervised learning and active learning are approaches which need a large pool of unlabelled data beneath a small pool of labelled data to work. Semi-supervised learning uses information in the unlabelled dataset to improve the supervised learning model by making assumptions about the data distribution. Zhu [20] distinguishes three kinds of semi-supervised learning in his literature survey: If samples from the same class are clustered into groups, clustering techniques like Gaussian mixture models can be used to get a prior for the probability of an unlabelled samples belonging to a certain class [21]. Self-learning is a technique in which an agent trains itself by assuming that samples it can classify with high confidence certainly have the predicted label and then adds them to the labelled set [22]. Co-training uses two classifiers, each trained on a different set of features, which teach it other with unlabelled samples they are highly confident about [23].

2.2. Active learning problem definition

There exist many different types, scenarios and setups of active learning and ways to formalize its objective. Furthermore, many different features can be used to choose which sample to be labelled next. The last subsection of this section gives an overview of the challenges of active learning.

2.2.1. Scenarios of active learning

There are three commonly used scenarios of active learning, which are also shown in Fig. 2.1:

- Membership query synthesis:
  In this scenario it is possible to generate a new sample in the input space of the model. This synthesis of a sample is only possible if the input space is clearly defined e.g. in a finite domain [24]. In many domains, however, synthesis samples can be very awkward to label by a human annotator. An example for this in the domain of handwritten digit classification would be a synthesized image showing a letter instead of a digit. More promising synthesizers in these domains are generative adversarial networks which have been used by Zhu and Bento [25].
- Pool-based:
  The pool-based scenario offers the active learner a huge pool of unlabelled data out of which one sample has to be chosen to be annotated next. The corresponding policy is given in Equation 2.1. The pool-based setup is a very general setup allowing to find the best sample to annotate per time step. It suffers from the large space of samples to choose from, making the decision computationally expensive and hard.

$$\begin{aligned} &pool\text{-}based: \; x_{next} = activeLearner(D_{unl}) \\ &with\ D_{unl} = unlabelled\ data\ pool = \{x_1, x_2, \dots x_n\} \\ &and\ x_{next} \subsetneq D\ being\ the\ sample\ to\ label\ next. \end{aligned} \tag{2.1}$$

- Stream-based:
  The last common scenario of active learning is the stream-based one: Its active learner receives one sample of the pool of unlabelled data at a time and only has to make a binary decision between querying the label of this sample or discarding it. The according policy is given in Equation 2.2. This setup has the disadvantage that it cannot choose the best point in the unlabelled dataset to query next, but rather chooses the next point being good enough. On the other hand it makes the decision much easier, as only one sample has to be evaluated and only a binary decision has to be made.

$$\begin{aligned} &stream\text{-}based\ policy: x_{current} \rightarrow \{query\ label, discard\} \\ &with\ x_{current}\ being\ the\ current\ instance\ of\ the\ data\ stream. \end{aligned} \tag{2.2}$$

The membership query synthesis scenario may also be called an interventional scenario and both the other scenarios selective scenarios, as done by Tong [26, p 5].

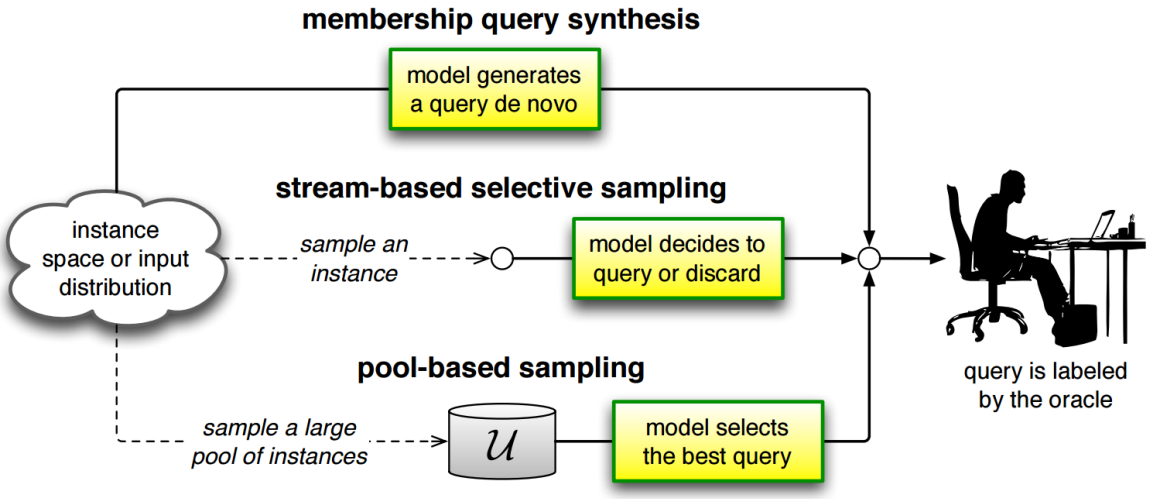

Figure 2.1.: Scenarios of active learning: New samples may be generated, chosen with a binary decision or chosen out of a pool of unlabelled samples.

Source: [1]

2.2.2. Annotation cost

There are many different variants how to calculate the annotation cost. These setups can be categorized by following aspects:

- Cost per query:
  In real-world cases there is usually a fixed cost for each query to the annotating oracle, independent of the number of samples per query. This cost must not necessarily be monetary, but it can also be the time delay caused by sending a sample to an oracle, waiting till it has time, letting it label it and sending the label back.

- Cost per sample:
  Most setups assume that the annotation cost per sample is fixed. However, in some domains, it depends on the sample itself. As an example, in text categorization tasks annotating a long document has higher costs than annotating a shorter one. The costs for annotating a specific sample may be known in advance, or they have to be estimated. Estimating sample-dependent costs not known in advance has been investigated by Settles et al. [27] and Arora et al. [28].

- Cost for acquiring additional features:
  In some domains it is possible for classifiers to invest effort to gather additional features. As an example, in medical contexts, it is possible to have additional examinations like a blood test or liver biopsy before diagnosing a disease. Greiner et al. [29] have set up an active classifier being able to decide when to invest in the collection of additional features depending on the currently known features and the certainty of the current diagnosis.

2.2.3. Sequential vs. batch-mode active learning

In the sequential active learning setup, the active learning agent always chooses one sample at a time to transfer to the oracle and query the label. This setup has the following disadvantages: As the cost for labelling is usually the sum of a cost per query and a cost per sample to annotate, asking to annotate multiple samples in one query reduces the average cost per sample. Furthermore, it is often far more practical to let multiple samples label in parallel, e.g. because the labelling is outsourced or the human experts annotating the data are more efficient when labelling multiple samples in a row.

This is the motivation for batch-mode active learning, which chooses a batch of samples to be labelled in each step. Besides the more efficient annotation, it has two main disadvantages: Firstly, choosing a batch instead of one sample makes the action space exponential with respect to the batch size, and thus the decision more difficult. Secondly the samples in a batch cannot profit from the model being trained on each other, while in the sequential setting, the model is re-trained after each sample. This can lead to very similar samples in one batch, because similar samples can be expected to improve the model equally well. One sample of them would be enough and additionally annotating the others is unnecessary, this decreases performance.

2.2.4. Objective of active learning

The goal of all active learners is to maximize the performance of the supervised learning model while minimizing annotation cost. As it is always possible to annotate more samples, which increases annotation cost, but usually allows to increase the model's performance, there clearly is a trade-off between these two goals. Most active learning tasks simplify this multi-objective problem to a single-objective problem by assuming that there is a fixed annotation budget and the only goal is to maximize the model performance till the annotation budget is exhausted. In contrast, Kapoor et al. [30] have rather taken misclassification costs into account and have set the objective of minimizing the sum of annotation and misclassification costs.

There are many different metrics which can be used to measure and formalize a supervised learning model's performance. Metrics like the accuracy, balanced error rate or F1-score can be used for classification problems and change significantly which samples should be chosen. As an example, one can assume that choosing samples from an underrepresented class is much more important if the balanced error rate instead of the unbalanced one should be minimized. Active learning for regression problems has only been covered by few papers so far, including Cai et al. [31]. The goal of most active learning agents for these problems is to choose samples based on the variance of the model prediction or the expected model change.

2.2.5. Observation space of active learner

The observation space is the basis on which the active learner decides. It consists of the unlabelled sample(s) and additional information calculated out of it. This information can be divided into following categories:

- Sample itself
  The sample itself can be used in the same way it is the input to the model trained.
- Encoded sample
  Some tasks, especially in natural language processing, prefer the samples to be encoded in some form, e.g. to ensure they have a fixed size or are language-independent.

- Similarity of sample to other samples:
  It can be helpful to choose samples, which are in dense regions of the underlying data distribution, because it can be expected that their information also affects many other samples. Furthermore, it can be helpful to choose samples which are not too close to the samples already annotated, as they would not add additional information in this case. Thus, the similarity of the samples to other samples in both the unlabelled data pool and annotated data pool, either to all samples, or just the maximum, minimum, and/or average similarity can be an important information for the active learning agent. In the batch-mode setup the similarity to the samples currently part of the batch is another information which can be used.

- Output prediction of model:
  The prediction of the model trained is usually the most important information to the learning agent, as it contains information about how sure the model is about the classification of this sample.
- Statistics of output prediction:
  While the output prediction contains the full information, statistics of it are widely used, especially by heuristics. An example would be the entropy of the prediction vector or its variance.
- Certainty of output prediction:
  Some machine learning models include an estimation of the uncertainty of their prediction, which is another useful feature.

- Costs of annotation:
  When the costs per sample differ between samples, the true or estimated annotation costs are also used as input to the active learner.
- Current use of annotation budget:
  The current use of the annotation budget might be used by the active learning agent as it might want to use different strategies in the beginning than in the end.

2.2.6. Challenges of active learning

Active learning has several challenges making it difficult to choose the best sample(s) to be labelled next, and thus making it hard to find well-working active learning agents.

- Challenges of low-resource settings:
  As active learning is only useful in low-resource settings, it is only used in such settings, and thus inherits all challenges which come up in such settings: There is a severe

risk of overfitting to the few training samples and it is hard to measure it because it is usually not possible to have a big validation set either. Thus, first a supervised learning model performing well in low-resource settings has to be developed before any active learning can be applied.

- High variance of supervised learning model:
  As the supervised learning model is only trained on very few samples in the beginning of the active learning process, there is a high variance of the supervised learning model, and thus also metrics dependent on the model, like the uncertainty of the prediction, have a high variance.
- Badly performing supervised learning model:
  Because of the small size of the training set, the supervised learning model often performs quite badly. Thus, the decision border of the current model might be far away from the border of the model trained on a large training set. Choosing samples always close to this border might change this border slightly but prevent the supervised learning model from finding a completely new decision border. Instead, the supervised learning model is stuck in a local minimum and all samples chosen by the active learning agent only help to find a slightly better point in the local minimum instead of searching globally for a better minimum.
- Low robustness and reliability:
  Prateek et al. [32] have shown that there are many problems in evaluating active learning agents reliably and reproducible. As an example, the reported performance of random sampling varies significantly between studies even given the same experimental conditions. Furthermore, they have shown that differences between different active learning agents may vanish when more advanced regularization techniques are used.

2.3. Heuristic active learning frameworks

Active learning frameworks are agents with a policy mapping from the observation, i.e. the information they have or can gain about the active learning process, to an action. Heuristic frameworks rely on engineered, fixed policies. Their performance depends highly on the application, with no framework being able to outperform the others in all cases. There are three core ideas which kind of samples should be chosen to be labelled:

- Choose uncertain samples changing the model
  The samples chosen by the active learning algorithm should be such that we gain a lot of information by knowing their label; thus, we should be unsure about their label. Various heuristics differ in how they define this uncertainty. There are also related methods, which try to estimate the model change or error reduction when adding a sample to the labelled set and choose the samples having the highest model change or error reduction.
  As the supervised learning model changes when labelling a sample, model-based heuristics usually perform worse in the batch-mode setting, when the supervised learning model is updated much more rarely.
- Choose representative samples:
  Outliers which are far away from other samples do not help to classify other samples,

thus, they should not be chosen. There are various distance metrics and methods to calculate the representativeness of a sample given the set of unlabelled samples.

- Choose diverse samples:
  While the samples chosen to be labelled should be representative of the unlabelled set, they should not be too close to samples the labelled set, as redundant samples do not add additional information. This is especially important if there are duplicates in the training set or many similar samples.

These ideas can be clearly distinguished, as they rely on mutually exclusive sets of features: The model-based approaches only calculate the 'usefulness' of a sample dependent on the supervised learning model. The representative-based approaches calculate it dependent on the other samples in the unlabelled set. The diversity-based approaches calculate it dependent on the samples in the labelled set.

To get an intuition of the strength and weakness of these ideas, consider the example active learning problem shown in Fig. 2.2. The task is to classify samples from two classes in a 2-dimensional space. The underlying, but unknown, data distribution is one Gaussian cloud for each class, located at (0,0) and (1,1) respectively. It is assumed that already 5 samples from each class are in the labelled set and a classifier (e.g. a support vector machine) draws a linear decision border (depicted in violet) between them. Because of the limited number of labelled samples, the decision border is not optimal, the optimal one would go through (0,1), (.5,.5) and (1,0). The active learning task is to choose the next data point to be labelled.
Candidate point A is directly on the decision border and far away from other labelled samples. Thus, an active learning heuristic relying on uncertainty and diversity sampling would probably choose it next. Nonetheless, the point is a poor choice, as it is an outlier and far away from other data points, it is not representative for the other unlabelled points.
Candidate point B is directly on the decision border and surrounded by many other unlabelled points. Thus, it is an uncertain and representative sample. Because it is very close to two already labelled points, it is also a poor choice.
Candidate point C is both very representative and has a minimum distance to already labelled points. Nonetheless, it is a poor choice, because it is far away from the decision border and cannot help to improve it. It is already nearly certain that it belongs to class 1.
One of the best points to label next is point D: It is close to the decision border, close to other unlabelled samples and therefore representative of them and has a decent distance to already labelled samples.
This example makes it clear that the best performance can be expected of agents combining model-based, diversity, and representative sampling, even though most heuristics rely on only one or two of these ideas.

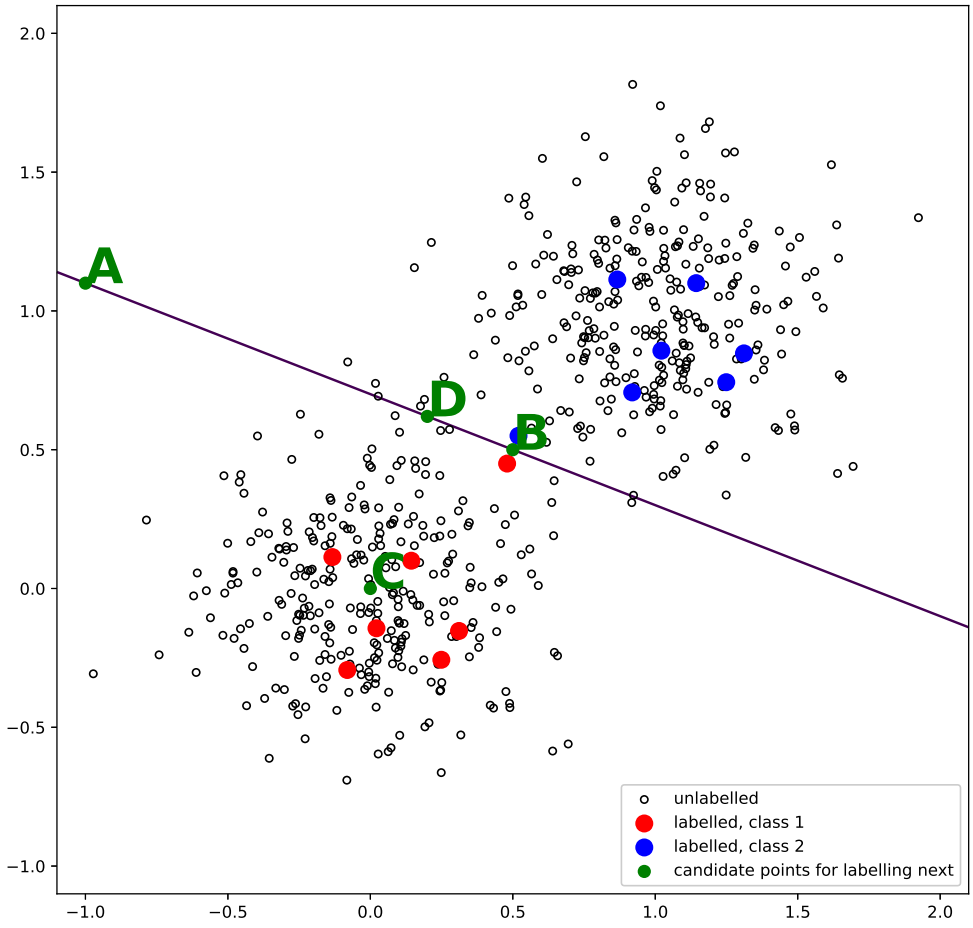

Figure 2.2.: Example of active learning problem: The candidate points to be labelled next differ a lot in their fulfilment of different criteria like uncertainty, diversity and representativeness.

2.3.1. Model-based / uncertainty based heuristics

Model-based heuristics calculate the 'usefulness' of a sample only given it and the supervised learning model trained on the labelled set.

- Uncertainty sampling:
  This heuristic chooses the next sample based only on the posterior probability of each class label under the supervised learning model. Its least confidence (LC) variant chooses the sample where the probability of the most likely label is smallest. [2] It has the disadvantage that the information about the probability of all other classes is discarded. This shortcoming is partly overcome by the margin sampling strategy, which

rather chooses the sample with the smallest difference in the likelihood of the most and second most likely sample. [33] However, the margin (M) approach still discards some information, which can cause problem in setting with a huge number of samples. In those settings the third variant of choosing the sample with the highest entropy (E) [34] of the prediction works well. The policies for all three variants are given in Equation 2.3.

$$
\begin{aligned}
x_{LC} &= \underset{x \in D_{unlabelled}}{\operatorname{argmin}} \; P(y_1|x) \\
x_{M} &= \underset{x \in D_{unlabelled}}{\operatorname{argmin}} \; P(y_1|x) - P(y_2|x) \\
x_{E} &= \underset{x \in D_{unlabelled}}{\operatorname{argmax}} \; Entropy(P(y|x)) \\
&\textit{with } x \textit{ being the sample to label next,} \\
&y_1 \textit{ being the most likely label,} \\
&y_2 \textit{ being the second most likely label,} \\
&Entropy(P(y|x)) = -\sum_i P(y_i|x) * ln(P(y_i|x)), \\
&\textit{and } P(y|x) \textit{ being the prediction of the supervised learning model on sample } x.
\end{aligned}
\tag{2.3}
$$

- Query-By-Committee:
  The query-by-committee (QBC) approach uses a set of models, that are all trained on the same labelled set, but differ significantly from each other. Then each committee member predicts the output for each sample in the unlabelled set. The next point to query is the point on which the committee members disagree most. To committee members may differ in supervised learning model (e.g. neural networks and SVMs), hyperparameters, starting points [35], partitioning of feature space [36] and ensemble learning methods like bagging and boosting [37]. Furthermore, there are various methods to measure disagreement including the vote entropy [35] and the sum of Kullback-Leibler (KL) divergences [38] between the prediction of each committee member and the average prediction of all committee members [39].
- Expected model change:
  This approach uses the sample which would change the supervised learning model most if its label was known and it was added to the labelled set. One strategy to measure the model change is to measure the gradient vector of the model when adding the sample, known as expected gradient length (EGL). As the true label is not known in advanced, the expectation over the possible labels given the current prediction of the model is taken. [40]
- Expected error reduction:
  This approach estimates the reduction of the model error for each sample if it is added to the labelled set. The error can be the misclassification ratio, log-loss, F1-measure or any other performance measure. It is calculated as error of the supervised learning model over the training set. As the true labels are not known, again the expectation over all possible labels is taken. All these steps make this approach computationally extremely costly, making it impractical in most scenarios. [1]

- Bayesian active learning by disagreement:

  Bayesian active learning by disagreement (BALD) was introduced by Houlsby et al. [41] and assumes that the classifier is a Gaussian process classifier. BALD aims to compute the entropy of a prediction of this classifier using several approximations. The final policy is to choose the point with the highest entropy.

  Computing this entropy is intractable for classifiers with a lot of parameters, like large neural networks. Thus, Gal and Ghahramani [42] proposed to use Monte Carlo dropout as Bayesian approximation: When using dropout also when predicting with a neural network, the prediction depends on the random choice of dropped neurons. When performing such random dropout several times, several different predictions are computed, whose variance can be seen as a measure of the uncertainty of the prediction. Gal et al. [43] treated this measure as measure for the 'usefulness' of choosing a sample to be labelled and presented a corresponding active learning agent.
  Chitta et al. used deep probabilistic ensembles as an approximation of Bayesian neural networks which scale better to larger datasets, models and annotations budgets. When interpreting the different models making up an ensemble as a committee of different models, this approach equals the query-by-committee heuristic from subsection 2.3.1.

2.3.2. Diversity based heuristics

Diversity sampling aims to choose samples which are dissimilar to already labelled samples. It is nearly always combined with uncertainty based sampling. There are many ways to combine heuristics, e.g. it is possible to first choose a subset of candidate points using one heuristic and then choose the final points to be labelled using the other heuristic, or one could directly define a heuristic combining two existing ones.

As an example, Wang et al. [44] have combined uncertainty sampling and diversity sampling by choosing the samples such that the sum of the uncertainty of the chosen samples and the diversity (as -1 * similarity) within them is maximized.

Batch-mode active learning heuristics profits a lot from diversity-sampling ensuring that diversity exists within the batch. They are described more closely in subsection 2.3.4.

2.3.3. Representativeness based heuristics

The methods explained so far have the disadvantage, that they might choose outliers. Density-weighted methods introduce a similarity measure, which measures how representative a sample is for the data distribution. They are nearly always combined with uncertainty based approaches: Samples having both a high informativeness (calculated using one of the frameworks above) and representativeness are chosen by choosing the sample maximizing a function of both, e.g. the product or a weighted sum. [1]

2.3.4. Heuristics for batch-mode active learning

Uncertainty-based heuristics have shown to have large performance drops when applied in the batch-mode active learning setup, which has led to the development of active learning heuristics specifically targeting the batch-mode setup.

Under the assumption that the Fisher information matrix [45] represents the uncertainty of a supervised learning model, which should be minimized, batch-mode active learning can be described as the problem of choosing the next batch to sample such that the ratio of the fisher information matrices of the model before and after adding the batch is minimized. This approach was used by Hoi et al. [10] (2006) to find a function calculating the expected model uncertainty reduction by labelling a batch. By analysing this function it becomes clear that there are three objectives for the samples in the batch:

- Uncertainty minimization:
  Samples with a large classification uncertainty should be chosen, similar to the uncertainty sampling heuristic.
- Dissimilarity to labelled set:
  Samples with a high similarity to labelled samples should not be chosen.
- Similarity to unlabelled set:
  By choosing only samples similar to the unlabelled set, the choice of outliers can be prohibited. This idea is used by density-weighted heuristic methods.

Thus, the approach of reducing the fisher information matrix can be seen as a non-engineered combination of heuristics, with a theoretical background for the choice of the combination. One huge disadvantage of this approach is the fact that the fisher information matrix can only be calculated efficiently for parametrised supervised learning models with very few parameters like logistic regression. It is neither applicable for large neural networks nor random forests.

Hoi et al. [46] (2009) also proposed another approach: They formulated batch-mode AL as a min-max optimization problem with the objective of choosing the samples minimizing the error of the supervised learning model even if the labels of the samples are such that they maximize the error. As this objective function is computationally very challenging, they relax in many ways including minimizing only its upper bound.

Demir et al. [47] proposed another architecture for batch-mode active learning: They chose promising samples using margin sampling for SVMs and then removed redundant samples based on distance metrics.

Sener and Savarese [14] describe active learning as a core-set-selection problem. Thus, they try to choose samples which are dissimilar both to each other and the labelled set and representative of the unlabelled set. They apply it to CNNs, as most active learning algorithms do not perform well for choosing samples for a CNN model. Different to all other heuristics, they do not use any model-based information. This has the advantage that their method does not need any labelled samples at all. Thus, its performance is both independent of the batch size and the starting set size.

Kirsch et al. [48] extended the active learning strategy of Bayesian Active Learning by Disagreement (BALD) such that it does not choose samples whose information overlap a lot.

They choose the samples to be added to the batch greedily one after another, which has the disadvantage that it is only a $1 - \frac{1}{e}$-approximate algorithm.

Liebgott et al. [49] have also shown that reducing the intra-batch redundancy further improves the performance of batch-mode active learning.

2.3.5. Shortcomings of heuristic active learning frameworks

The shortcoming of heuristic active learning frameworks are the following:

- Not combining various ideas:
  The heuristic active learning frameworks do not combine the various ideas outlined in the enumeration 2.3 which samples should be chosen. Even if they are combined, the choice how they are combined (additive, multiplicative, more complex ...) is another engineered heuristic.
- Not adaptive to supervised learning model:
  Heuristics do not change their policy based on the supervised learning model used, which may have specific advantages or weaknesses.
- Not adaptive to dataset:
  Heuristics keep their policy irrespective of the dataset, which may be imbalanced, contain many or only very few outliers or have a very high variance of the quality of the samples. Heuristics cannot adapt to all these characteristics of a dataset.
- Not adaptive to metric to optimize:
  There are many metrics in supervised learning one can optimize including the loss, accuracy, F1-score, balanced error rate or intersection over union score. Depending on the score different active learning strategies are best. Heuristics cannot adapt to different metrics to optimize.

Because of these shortcomings, the performance of active learning methods varies wildly between various tasks, datasets, supervised learning models and metrics and there is no heuristic known to perform good in all cases.

2.4. Learning active learning

Learning active learning methods try to overcome the shortcomings of heuristics by learning the optimal policy. They learn a policy mapping from many different features to the sample maximizing a certain metric for a specific supervised learning model and dataset. If such a learned policy is then applied on task with the same features, metric to optimize, supervised learning model and a similar dataset, it may outperform engineered metrics as it has learned to use all features available and adapt to the metric and supervised learning model. To define the objective of the learning agent, it is useful to describe active learning as a Markov Decision Process (MDP). Current learning active learning frameworks mostly use the pool-based scenario and the sequential active learning setup, defining parts of the action and state space of the MDP.

2.4.1. Simplification of MDP for active learning

Most approaches currently used in literature all try to choose the next sample such that the performance of the supervised learning model after labelling this sample is maximized. They ignore the fact that it may be possible to choose actions with lower earlier rewards to have even higher later rewards. While this may reduce the performance at the end of the active learning process, it has two advantages: First it promotes anytime behaviour, ensuring that the active learning performed best even if it was terminated early, which was already pointed out by Bachmann et al. [5]. Second it makes it much easier to implement and to train the active learning agents. In terminology of Markov decision processes, this can be described as choosing a discount factor of 0, which sets the objective of maximizing the reward in the next step independent of any later rewards.

2.4.2. Pipeline for training active learning frameworks

The pipeline for training most learning active learning frameworks in current literature is split into the generation of training data and the training of an agent on these data. First the training data is generated and saved, then the agent is trained on the saved data. This has the advantage that different agents can be tried on the same generated data or their hyperparameters may be optimized without needing perform the computationally expensive generation of training data again.

The purpose of the generation of training data is to generate many (observation, action, reward)-tuples, which act as training data for the learning agent. The observation, action and reward space are described more closely in subsection 3.1.1. The choice of the policy used while training the samples is one important design choice of this step. In current literature following policies are used:

- random sampling (off-policy)
- sampling with algorithmic expert (off-policy)
- sampling with the learning agent trained on the past data (on-policy)
- mixture of these policies (e.g. 50% random sampling, 50% on-policy)

On-policy training is used by Fang et al [3] and Bachmann et al. [5]. Konyushkova et al. [6] implemented both on-policy and off-policy training and Liu et al. [50] use a mixture of training with the current agent and the algorithmic expert which equals mixing on-policy and off-policy training.
In each step of the active learning process not only the model improvement by the chosen sample can be calculated, but also the model improvement when another sample would be labelled. This equals evaluating the reward function R(s,a) of a Markov decision process without actually performing the step. Both Liu et al. [50] and Konyushkova et al. [6] use this approach and calculate the model improvement for different samples if this sample would be added to the labelled set, Liu et al. call this an algorithmic expert. Besides allowing the generation of more training data, the concept of the algorithmic expert also has another advantage: When used as an active learning policy it provides a very strong benchmark for the maximum performance achievable if the active learner would already know the labels of the unlabelled set and could choose the best sample with this additional knowledge. Thus, it

provides an upper bound for the performance of active learning agents.
Calculating the model improvement by the algorithmic expert is computationally expensive as it includes a retraining of the model for each unlabelled sample, thus both Liu et al. [50] and Konyushkova et al. [6] only calculate it for a small number of samples in the unlabelled set.

2.4.3. Reinforcement learning for MDPs

As the MDP of active learning is unknown for new tasks and dataset, methods to find the optimal policy given this MDP must rely on data. Thus, dynamic programming is not feasible, instead reinforcement learning algorithms are the suitable method for this problem. They can be divided into model-based and model-free algorithms [51]: Model-based algorithms try to estimate the unknown MDP so that they can plan their next steps. This makes them very sample-efficient, but computationally expensive, especially if the state and action space are very big. Model-free algorithms instead only try to learn state/action-values and/or policies without estimating the MDP. In the case of active learning, the state and action space are very big, while there is no cost associated with sampling. Thus, model-free algorithms are more suitable.

In the sequential active learning setting, the action space is a categorical one-dimensional space allowing to efficiently calculate the action maximizing a function given the action. This makes the common reinforcement learning framework of Q-Learning suitable. It fits a regression function Q(observation, action) to the training data, which tries to predict the reward given the observation for each action. Its policy $\pi(observation)$ is simply choosing the action maximizing the Q-Value given this observation. This approach is used by Konyushkova et al. [6], Bachmann et al. [5] and Fang et al. [3]. Another approach is used by Liu et al. [50], [52]: They use imitation learning and train the agent to choose the action maximizing the reward rather than to predict the reward. While these approaches differ conceptually, they are very similar concerning the implementation: Both try to find a function mapping from the observation to one scalar value per action, which is a classical supervised learning problem. For Q-Learning the target is directly the reward of this action, for imitation learning the target is one if the reward is the maximum reward achievable, else it is zero.

2.4.4. Challenges of learning active learning

Learning active learning includes three subproblems of machine learning:

- Active learning:
  Learning active learning includes all challenges of active learning, which were already described in subsection 2.2.6.
- Reinforcement learning:
  Finding an optimal policy of a black-box MDP is a reinforcement learning problem, thus learning active learning also has to cope with the credit assignment problem and fit a supervised learning model on the value- and/or policy function. Furthermore, the optimal trade-off between exploration and exploitation hast to be determined. As reinforcement learning problem it is also likely, that problems with robustness and reliability are encountered.

- Transfer learning:
  As the learning agent is trained on one task and applied on another, it is essentially performing transfer learning, and thus has to cope with the challenges of transfer learning. As an example, one must prevent the learning agent to overfit on the task it was trained on and perform well on another task where the scale and distribution of features may differ a lot.

Solving these three subproblems at once including their interaction with each other makes learning active learning very challenging. There is a huge space of hyperparameters and design choices which can be explored and it was found that it is quite hard to find a well-working set of them.

Exploring the whole set of hyperparameters is not possible for two reasons: Firstly, generating the training data for the learning agent is computationally very expensive, as a supervised learning model has to be trained for every single data point generated. Second, there is a very high variance in the performance of an active learning agent on a task, dependent on the randomly sampled starting set. Thus, many repetitions are needed to find out whether a change of hyperparameters or design choices changed the performance significantly.

2.4.5. Learning batch-mode active learning

The learning agents trained on sequential batch-mode active learning do not perform well on batch-mode active learning as the supervised learning model is not updated after each step anymore. To ensure that there is a high diversity within the batch, metrics like the similarity of a sample to the samples in the batch can be added to the observation space.

Learning batch-mode active learning has partly been covered by Ravi and Larochell [53]: Similar to Bachmann et al., Konyushkova et al. and Liu et al. they train a regression model to predict the improvement of the accuracy if a sample is added to the labelled set. Then they multiply this quality metric with a diversity metric to gain a final expert metric. The diversity metric takes the similarity of a sample to other unlabelled samples into account. The policy is to choose the sample(s) having the highest expert metric. While the quality metric is a learned metric, it is a design choice to maximize the product of the quality metric and the diversity metric. Thus, this approach can be seen as a mixture of an engineered heuristic and a learning agent.

There is no known literature on describing batch-mode active learning as a Markov decision process and solving it with learning methods, which is a gap this thesis tries to fill.

3. Framework for learning batch-mode active learning

Our framework for learning active learning in the batch-mode setting is made up of three components:

The active learning task is the problem on which an active learning agent is applied. It consists of a dataset on which a classifier is trained on, the supervised learning model for the classification and an active learning environment describing active learning given such a task as a Markov decision process (MDP). This formalization of active learning is described in the first section of this chapter.

The next section describes how one episode of active learning is made up: It consists of the application of an active learning agent on an active learning problem described as MDP. The episodes ends when the annotation budget is exhausted. Such applications of an agent on an MDP are needed both to train a learning active learning agent and to evaluate its performance.

The last component of the framework are the agents learning active learning. We tried three different approaches, namely one approach using Q-Learning, one using Monte Carlo Policy search and one using uncertainty-weighted clustering. These agents are described in a separate section each.

3.1. Formalization of active learning as Markov decision process

By describing active learning as a Markov Decision Process (MDP) it becomes a well-known problem, for which many methods exist. A MDP is a tuple of the state space S, action space A, transition function and reward function R. The transition function is a function mapping from a state and action to a new state and a reward. An agent acting in this environment has in general a policy assigning a probability to the choice of an action given the state. The objective is to maximize is the cumulative reward, which is the sum of rewards obtained in each step. Often later rewards are discounted to prevent an infinite sum and to facilitate the convergence of some reinforcement learning algorithms. The corresponding equations are given in a formalized form in Equation 3.1.

$$
\begin{aligned}
&transition function : S \times A \rightarrow S \times \mathbb{R} \\
&agent's\ policy\ \Pi(a|s) : S \times A \rightarrow [0;\ 1] \\
&cumulative\ reward :\ R = \sum_{t=1}^{\infty} \gamma^t * r_t \\
&with\ discount\ factor\ \gamma \in (0;\ 1) \\
&and\ r_t\ =\ reward\ at\ step\ t \\
&and\ r_t \in \mathbb{R}, a \in A, s \in S.
\end{aligned}
\tag{3.1}
$$

3.1.1. Markov decision process for sequential active learning

Before defining the Markov decision process for batch-mode active learning, we first define it for sequential active learning. There are two reasons for this: First, sequential active learning is also implemented in this thesis as comparative benchmark. Second, describing both setups make it easier to show how the sequential batch-filling active learning we propose is defined. The MDP for sequential active learning with constant annotation costs and a fixed annotation budget has following components:

- Definition of episode:
  An episode is a sequence of multiple actions taken by the agent. It ends when the annotation budget is exhausted.
- State space:
  A state space consisting of the labelled and unlabelled set of samples and the model trained on the labelled set would include all information about the current state of the MDP completely. As this space is extremely huge and can include millions of variables, it is very hard for an agent to learn from this state. Thus, we define an observation space, which includes features of the state which might be relevant to the active learning agent.
- Observation space:
  The observation space containing the information on whose basis the active learner decides, is described in subsection 2.2.5. Most heuristics and algorithms only use a small subset of this space for their decision.
- Action space:
  An action or decision is the categorical choice of one sample in the unlabelled set. As the unlabelled set becomes smaller each step, the action space is variable.
- Reward function:
  We set the objective to maximize the model accuracy within a fixed annotation budget, which is the most common active learning objective. As this reward is very delayed, the credit assignment problem occurs: It becomes very hard for the agent to find out, whether a specific action was good or bad. This problem was made popular by Minsky in 1961 [54]. To circumvent this problem reward shaping is used [55]: Instead, of only returning a reward for the model accuracy in the end, the reward is rather the

improvement of the current model's accuracy in each step, according to Equation 3.2.

$$reward(labelling\ x_t) = modelAccuracy(D_{labelled} \cup \{x_t\}) - modelAccuracy(D_{labelled}) \tag{3.2}$$

As the sum of all rewards per step equals the total accuracy improvement over all episodes, the objective stays the same, but the learning process is faster. If the supervised learning model allows to use smoother performance metrics than the accuracy, e.g. a loss, the reward can be set to be the reduction of the loss instead of the improvement of the accuracy.

- Transition function:
  The last component of a MDP is the transition function mapping the current state and action probabilistically to the reward, a new state and a new observation fully defined by the new state. In the case of active learning this step includes the annotation of a chosen sample, the re-training of the model with it, the change of both the labelled and unlabelled dataset, and the increase of the currently spent annotation budget. If it is assumed that the state is an internal variable of the Markov decision process, a step function can be defined as externally available representation of the transition function. It is assumed that the function gets a sample to label (i.e. the action) as input, performs the update step by changing the internal state and then returns the new observation, the reward and whether the episode ended or not. The pseudocode in Algorithm 1 can be used for this step function:

Algorithm 1 Pseudocode of the step function of a sequential pool-based active learning MDP

Require: $task$, $D_{unlabelled}$, $D_{labelled}$, $annotationBudget$, $oldAccuracy$
  function $stepFunction(sampleToLabel)$
    $D_{unlabelled} \leftarrow D_{unlabelled} \setminus \{sampleToLabel\}$
    $D_{labelled} \leftarrow D_{labelled} \cup \{sampleToLabel\}$
    $newAccuracy \leftarrow task.trainOn(D_{labelled})$
    $reward \leftarrow newAccuracy - oldAccuracy$
    $oldAccuracy \leftarrow newAccuracy$
    $observation \leftarrow task.getObservation(D_{unlabelled}, D_{labelled})$
    $epochFinished \leftarrow |D_{labelled}| >= annotationBudget$
    return $observation$, $reward$, $epochFinished$
  end function

3.1.2. Markov decision process for sequential batch-filling active learning

Choosing a batch of samples to be annotated next is a very hard task: If one wants to choose b samples out of a pool of n unlabelled samples, there are $\frac{n!}{(n-b)!}$ options. Thus, we propose to change the definition of a step of the MDP to be the choice of one sample which is added to the batch. The corresponding pseudocode for this sequential batch-filling MDP is shown in Algorithm 2. First the active learning agent chooses the first sample to add to the batch based on the observation defined before in subsection 2.2.5. Then the next sample is chosen based on the same information and additional measures for the similarity of each unlabelled

sample to the sample(s) currently in the batch. This way the agent can choose samples which are not too close to samples already in the batch. While the unlabelled set becomes smaller every step as one sample leaves it and enters the batch, the labelled set is only updated if the batch is full (i.e. the if-condition in the pseudocode is fulfilled): Then the labelled set is extended by all samples in the batch, thus all samples in the batch are annotated. As the labelled set was updated, the supervised learning model can be retrained on it, changing the observation significantly. The components of the MDP stay largely the same compared to sequential active learning: The definition of an episode and the action space stay the same. The state space is extended by the set of samples in the batch, the observation space is extended by the similarities of the unlabelled samples to the samples in the batch. The transition function still has the same input and return values. Thus, any agents applied on sequential batch-filling active learning stay mostly the same, they only have to adapt to the extended observation space. Most changes occur within the transition function: The labelled set, trained supervised learning model and reward value are only updated if the batch is full.

Algorithm 2 Pseudocode of the step function of a batch-mode pool-based active learning MDP

Require: $task$, $D_{unlabelled}$, $D_{labelled}$, $annotationBudget$, $oldAccuracy$, $batchSize$
  function $stepFunction(sampleToLabel)$
    $D_{unlabelled} \leftarrow D_{unlabelled} \setminus \{sampleToLabel\}$
    $batch \leftarrow batch \cup \{sampleToLabel\}$
    if $|batch| == batchSize$ **then**
      $D_{labelled} \leftarrow D_{labelled} \cup batch$
      $batch \leftarrow \{\}$
      $newAccuracy \leftarrow task.trainOn(D_{labelled})$
      $reward \leftarrow newAccuracy - oldAccuracy$
      $oldAccuracy \leftarrow newAccuracy$
    else
      $reward \leftarrow 0$
    end if
    $observation \leftarrow task.getObservation(D_{unlabelled}, D_{labelled}, batch)$
    $epochFinished \leftarrow |D_{labelled}| + |batch| >= annotationBudget$
    return $observation$, $reward$, $epochFinished$
  end function

This definition of sequential batch-filling active learning combines the advantages of sequential and batch-mode active learning. The action per step is still one categorical choice like for sequential active learning, making it much easier for a learning agent to learn which sample to choose next. As the labelling is only performed for a full batch, the practical advantages of batch-mode learning are still preserved. However, there is a theoretical disadvantage: Compared to true sequential learning, the supervised learning model is not trained on the samples just chosen, and thus the observation space is not updated. Thus, it can be expected, that a sequential batch-filling agent performs slightly worse than a true sequential agent, with the difference depending on the batch size and agent.

3.1.3. Observation space

The observation space consists of many features for each sample in a specific step of the active learning process. The features can be divided into two groups:

The first group of features is specific for each sample, it is computed for each sample in the unlabelled set:

- entropy of the prediction
- uncertainty of the prediction
- percentiles of euclidean distances of the prediction of the sample to the predictions of the labelled and batch set
- percentiles of euclidean distances of the representation of the sample to the representations of the labelled and batch set
- cosine similarity, Manhattan distance and euclidean distance of the sample's prediction to the mean predictions of the labelled and batch set and the unlabelled set
- cosine similarity, Manhattan distance and euclidean distance of the sample's representation to the mean representations of the labelled and batch set and the unlabelled set

The percentiles of the distances are used to ensure that the size of the features is independent of the number of samples in the labelled and batch set and the unlabelled set. For the computation of features, always the union of the labelled set and batch set, rather than any set itself is used, as samples which are already in the batch are already chosen for labelling, thus they do not need to be treated differently than labelled samples. To be able to compute distances between samples efficiently, we decided to use a one-dimensional representation of each sample for the distance calculations. We use the raw sample itself as representation for datasets having unstructured data. Image samples are first broken down to 200 features using a principal component analysis [56] and then to 3 features using tSNE embeddings [57]. Natural language sentences and answers are represented by their word2vec embeddings [58]. The prediction itself is not used in its raw form, as the size of the prediction is dependent on the number of classes in the active learning task. Using the raw form would prevent to use an agent trained on one task on a task with a different number of classes.

The second group only consists of features which are the same for all samples, as they describe the status of the active learning process:

- current number of samples in the labelled set
- current relative use of the annotation budget
- mean values of all features in the first group (mean across the samples)

The learning agents do not need to use all of these features.

3.2. Application of agent on active learning problem

The application of an agent on an active learning problem needs three components: First the task and dataset on which the active learning is performed needs to be defined. Next a

Markov decision process has to be defined given the task. It contains a reset function returning the initial observation and a step function taking an action as argument and returning the new observation, the reward and whether the episode has finished. The agents need a policy mapping from the observation to the next action. The pseudocode for combining these components is the following:

Algorithm 3 Pseudocode for applying agent on active learning MDP generated from task

Require: *agent*, *task*, *MDP*
  $mdp \leftarrow MDP(task)$
  $observation \leftarrow mdp.reset()$
  $epochFinished \leftarrow False$
  while *not* $epochFinished$ **do**
    $action \leftarrow agent.policy(observation)$
    $observation, reward, epochFinished \leftarrow mdp.step(action)$
  end while

3.3. Approach 1: Q-Learning

The purpose of training a learning agent is finding an optimal policy given a set of generated (observation, action, reward)-tuples. Many reinforcement learning frameworks are suitable for such problems. As already explained in subsection 2.4.1, model-free frameworks are most suitable for the active learning MDP. As the action space is a categorical one-dimensional space, finding the action maximizing an action-value function is very easy: One can simply try all actions and then choose the best one. Thus, Q-learning is a suitable approach for this problem. The Q-function and the corresponding policy are given in Equation 3.3. The goal is to find the optimal policy Π^* given the Q-function, while the Q-function itself depends on the policy. Thus an iterative approach is necessary to co-adapt the policy and the corresponding Q-function till the optimal policy is found.

$$\begin{aligned} Q(s,a) &= \mathbb{E}[R|s,a,\Pi] \\ \Pi^* &= argmax_\Pi\, \mathbb{E}[R] \\ &= argmax_\Pi\, Q(s,a). \end{aligned} \tag{3.3}$$

Estimating the reward given the state and action, i.e. estimating the Q-function has two parts: First a large set of training data, i.e. (reward, action, state)-samples has to be generated. In a second step, the Q-function is fitted on the training data to predict the reward given the state and action. As described in subsection 2.4.1, we use the simplification of a discount factor of zero, thus the reward to maximize is the reward in the next step. The generation of training data and the policy used for it is described in the next two subsections of this section, the fitting of the Q-function to the training data in the last one.

Alternatively to Q-learning we also tried to find an optimal policy by using imitation learning, which tried to predict the best action in a given state directly (i.e. a target of 1 for the best action, else 0), instead of having the reward as target. As it performed worse than Q-learning, we did not use it further.

3.3.1. Generation of training data

The purpose of this step is to generate many (reward, action, state)-samples a learning agent can be trained on. We designed the procedure for this step such that it fulfils three requirements:

- Parallelization:
  In order to speed up the training on servers with many cores, it should be possible to run multiple epochs in parallel.

- On-policy training possible:
  It should be possible to train the learning agent on-policy. This means, that the agent is trained on data which was gained using the agent's policy. A counter example would be an agent trained on data which was gained using a random agent, this is called off-policy training.
  We mix an on-policy agent with a random agent to use an ε-greedy policy during training. The ε-greedy policy is a commonly used policy for training reinforcement learning agents [12, p. 124ff]. It tries to solve the exploration-exploitation tradeoff in reinforcement learning by choosing in each state a random action with probability ε and else the best action according to the current policy or value function with probability $1 - \varepsilon$. By decreasing ε over time, this policy starts with a very explorative random policy and converges towards an on-policy training with the policy trained on past data. We use the ε-greedy policy and decrease the ε-value linearly from one to zero.

- Training of agents on all data at once:
  Supervised learning agents not relying on gradients, like random forests for example, do not support to train them iteratively on generated training samples. Thus, the procedure should allow to train a learning agent on the whole set of generated samples at once. Training on all data at once also makes the training faster, as the overhead is reduced: It is faster to train once on n training samples, than to train n times on one training sample each.

The algorithm we use for the generation of training data, which fulfils all these requirements, is shown in Algorithm 4. In the beginning, the training samples are generated with an agent using random sampling as policy. After each generation of training samples, a new learning agent is trained on these samples. The agent to use in the next iteration is an ϵ-greedy or mixed agent, combining an agent using random sampling and the learning agent just trained. The probability of each agent is determined by the ϵ-parameter which decreases with each iteration.

Algorithm 4 Top-level pseudocode for the generation of training data.

Require: $ApplicationHandler, al\_MDP, Agent\_randomSampling, Agent\_learning, Agent\_mixed, \varepsilon\_policy, noIterations, noEpisodesInParallel$

```
agent ← Agent_randomSampling()
applicationHandler ← ApplicationHandler(al_MDP)
for i ← 1 to noIterations do
    trainingSamples ← parallelize(applicationHandler.runEpisode(agent), noEpisodesInParallel)
    agent_learning ← Agent_learning().train(trainingSamples)
    agent ← Agent_mixed(Agent_randomSampling(), agent_learning, ε_policy(i))
end for
return trainingSamples, agent_learning
```

Apart from the advantages of parallel data generation and efficient training, this setup also has an important disadvantage: The learning agent is only updated once per iteration instead of instantly on newly generate data. Thus, the agent does not perform exactly on-policy, but is always trained on training samples generated by a previous version of itself.

3.3.2. Generation of training data with algorithmic expert

In reinforcement learning, usually one training sample is generated per step of the MDP. As each step of the active learning MDP needs to retrain a supervised learning model and perform a prediction for each sample in the unlabelled set, it is very costly to generate training samples. To soften this problem, Liu et al. [50] and Konyushkova et al [6] have used a so-called Algorithmic Expert to train active learning agents. In a given state of the active learning MDP, it calculates the model improvement for each sample in an unlabelled set if this sample would be added to the labelled set. This equals evaluating the reward function of the MDP for sequential active learning. It has the advantage that more (reward, action, state)-samples can be generated in a given state without needing to actually perform the update step.

The pseudocode for calculating the improvement of the loss for every sample using this approach is shown in Algorithm 5.

Algorithm 5 Pseudocode for classical algorithmic expert following Liu et al.

Require: $model, D_{unlabelled}, D_{validation}, oracle, currentLoss$

function $getImprovement(sample)$
  $trueLabel \leftarrow oracle(sample)$
  $oldModel \leftarrow model$
  $model.trainOn(sample, trueLabel)$
  $newLoss \leftarrow model.evaluate(D_{validation})$
  $model \leftarrow oldModel$
  $improvement \leftarrow currentLoss - newLoss$
  return $improvement$
end function

for $i \leftarrow 1$ *to* $|D_{unlabelled}|$ **do**
  $sample \leftarrow D_{unlabelled}.getSampleByIndex(i)$
  $improvements[i] \leftarrow getImprovement(sample)$
end for

As not the absolute improvement per sample but rather the relative improvement when labelling one sample compared to labelling other samples is needed to teach the learning agents to choose good actions, it is not necessary to train the model fully on a sample. Instead, for neural networks, it is sufficient to train it only one epoch on only this sample. This needs much fewer computations per sample than one step of the top-level MDP, where the model is retrained multiple epochs on the whole labelled set. Thus, the basic idea behind the algorithmic expert by Liu et al. promises a faster generation of training information for the learning agents.

Their variant of the algorithmic expert needs one training step and one evaluation step for each sample in the unlabelled dataset, thus the training complexity is $O(n)$ and the evaluation complexity even $O(n * m)$ with $n = |D_{unlabelled}|$ and $m = |D_{validation}|$. Because of these high computational costs, the improvement is only calculated on a subset of the unlabelled dataset, usually 5 to 10 samples, making the algorithmic expert weaker.

In order to develop a computationally cheaper algorithmic expert, we made the following assumption: “The improvement of the prediction on a dataset A when training on a dataset B is highly positively correlated to the improvement of the prediction on dataset B when training on dataset A” . The classical algorithmic expert calculates the improvement on the validation set (= dataset A) when training on a sample (= dataset B). According to the assumption the improvement on the sample (= dataset B) when training on the validation set (= dataset A) is highly correlated to the improvement as calculated by Liu et al. Thus, the training and validation set can be swapped. This assumption allows to estimate the improvement with a much lower complexity with Algorithm 6.

Algorithm 6 Pseudocode for adapted algorithmic expert

Require: $model, D_{unlabelled}, D_{validation}, oracle$
$trueLabels \leftarrow oracle(D_{unlabelled})$
$oldLosses \leftarrow model.evaluateSampleWise(D_{unlabelled}, trueLabels)$
$oldModel \leftarrow model$
$model.trainOn(D_{validation})$
$newLosses \leftarrow model.evaluateSampleWise(D_{unlabelled}, trueLabels)$
$model \leftarrow oldModel$
$improvements \leftarrow oldLosses - newLosses$

As no loop is necessary anymore, the training and evaluation complexity are reduced to $O(m)$ and $O(n)$ respectively, which is much faster if $1 << m << n$. Furthermore, the number of calls of the training and evaluation function is only $O(1)$ instead of $O(n)$ which makes the algorithmic expert up to n times faster if not the computations itself but the overhead caused by the function calls are limiting the performance.

Empirical evaluations of this adapted algorithmic expert have three results:

1. Performance when used directly as agent:
   The adapted algorithmic expert shows extremely good performance when used to choose the samples maximizing the accuracy. It even outperforms the classical algorithmic expert (evaluating 10 samples and choosing the best among them) by far, at it is able to evaluate the improvement for all samples in the unlabelled set in the same time, and thus can choose the best sample among all of them.

2. Correlation to improvements of classical algorithmic expert:
   The improvements of the adapted and classical algorithmic expert have shown to be only loosely correlated, with correlation factors between 0.1 and 0.3, depending on the task used to generate the data and the loss function used. Thus, the assumption made before in 3.3.2 is false.

3. Usability for generating training data for learning active learning agents:
   The regression models trained to predict the improvements of the model, as calculated by the adapted algorithmic expert, were not able to learn a good model. The mean squared error (MSE) loss achieved was 0.8 (while the targets were normalized to have unit variance). Thus, the regression models could only explain 20% of the variance of the improvements calculated by the adapted algorithmic expert. However, the regression models could explain more than 80% of the variance (MSE of 0.2) of the improvements calculated by classical algorithmic expert. The same results were found when not using the own code and tasks, but using the code and tasks with machine learning models by Konyushkova et al [6]: It was much harder to predict the improvements calculated by the adapted algorithmic expert than by the classical algorithmic expert.
   Consequently the adapted algorithmic expert is not usable to train learning agents and was not used further.

3.3.3. Fitting of Q-function

We designed the Q-function such that its task is not to predict the reward for all actions at once given the observation, but rather to predict the reward for one specific action (i.e. unlabelled sample) given the observation and information about this action. Thus, we are using the right design option as shown in Fig. 3.1, instead of the left. As this Q-function is applied sample-wise, its input space can be designed to be much smaller: Not all information about all samples in the unlabelled set is part of it, but only the information for one specific unlabelled sample/action. As an example: if the observation space consist of f_i sample-independent features and f_d features per each of n samples, then the left design would have $f_i + n * f_d$ input features, but the right one only $f_i + f_d$. Thus, the Q-function can be smaller and already encodes by design the assumption that the features of one sample are most important for estimating the value of this sample, which makes the fitting of it much easier. The disadvantage of the right architecture is that n Q-function evaluations, one for each sample, are necessary to find the optimal action in a state instead of only one. As the evaluations can be easily vectorized and are thus quite cheap, this is only a tiny disadvantage.

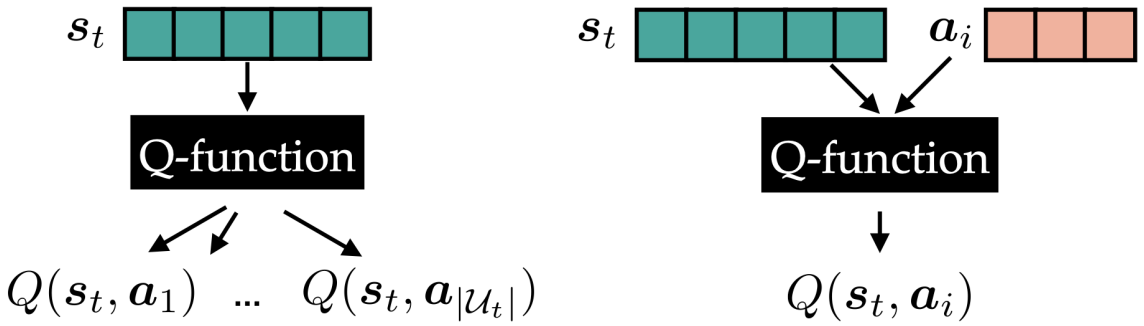

Figure 3.1.: Architecture options of Q-function: The action may either be encoded in the output, or be part of the input.

Source: [59]

We tried three different supervised learning model for fitting the Q-function: A deep neural network minimizing the mean square loss, an ordinary least squares regressor and a random forest. Both the deep neural network and ordinary least squares regressor perform badly, their MSE loss is only slightly smaller than the variance of the rewards, i.e. they perform about as good as a mean predictor. Thus, only the random forest was used in the following.

The hyperparameters of it, namely the number of estimators, the maximum depth and the minimum number of samples to split were optimized using automatic hyperparameter optimization with tree parzen estimators [60]. One challenge in this process was to determine the optimal hyperparameters for the transfer learning task of training the agent on samples from one task and then applying it on another task. It was found that the learning agent overfits on one task if both the training set and validation set come from the same task. Thus, we set the objective of the hyperparameter optimization to minimize the MSE loss on data generated on one task when using training data from another task. This way it was ensured that the hyperparameters maximize the transfer learning performance.

3.4. Approach 2: Monte Carlo policy search

Because of problems with the Q-learning approach, especially the low reproducibility, we also tried a different approach to learn active learning: We designed an active learning policy to combine the features of different heuristics and trained it using Monte Carlo policy search. The basic idea is to have several different heuristics which all assign a measure for the usefulness of choosing a certain data point to be labelled next and each choose the data point with the highest usefulness. agent combining different heuristics chooses the sample having the highest overall usefulness defined as the weighted sum of the usefulness measure of each heuristic. The learning takes place when finding the best weights: Monte Carlo policy search is used to find them, and thus determines the importance of each heuristic relative to the others. This agent can be interpreted in two different ways: On the one hand, it is a parametrised policy, whose parameters are learned by Monte Carlo policy search, thus it is a reinforcement learning approach. On the other hand, it is an ensemble of several different heuristics, so again a heuristic. The weights of the heuristics are just hyperparameters, which are learned through an hyperparameter optimization algorithm.

3.4.1. Linear random policy

When treating the usefulness computed by a heuristic as feature and the weights as linear factor, the ensemble active learning agent computes a usefulness value for each sample dependent on the features. It then chooses the sample with the highest value to be added next to the batch. While many different heuristics can be used, we found that using an ensemble of random sampling, entropy-based uncertainty sampling, and distance-based diversity and representativeness sampling performed best. We also tried to include the prediction variance measure used by Bayesian active learning by disagreement (BALD), but it did not improve the performance while increasing the computational cost a lot. The function we finally used to compute the usefulness of a sample is given in Equation 3.4.

$$\begin{aligned} \text{value} \leftarrow \epsilon &+ \beta_1 * \text{entropy of sample prediction} \\ &+ \beta_2 * \text{minimum distance to labelled and batch samples (0th percentile)} \\ &+ \beta_3 * \text{10th percentile of distances to labelled and batch samples} \\ &+ \beta_4 * \text{5th percentile of distances to unlabelled samples} \\ &+ \beta_5 * \text{10th percentile of distances to unlabelled samples} \\ &+ \beta_6 * \text{20th percentile of distances to unlabelled samples} \\ &\textit{with } \epsilon \sim N(\mu = 0, \sigma = 1). \end{aligned} \tag{3.4}$$

These features capture the three main ideas of active learning from the enumeration 2.3 and corresponding heuristics: The entropy captures the uncertainty of the prediction of a sample, the distances to the labelled and batch set allow to choose samples not too similar to ones already labelled or chosen for labelling and the distance to the unlabelled samples captures how representative a sample is for the dataset. We also tried to to learn a policy with the full feature set defined before as the observation space in subsection 3.1.3, and with more distance percentiles than just the two to three, but it even performed worse. An explanation for this

is given by the covariance matrix of the features, which was found to be very similar across different datasets and starting sizes: The three groups of features (prediction entropy, distance percentiles to labelled set + batch, distance percentiles to unlabelled set) are only slightly correlated to each other (correlation coefficients < 0.2), while the correlation coefficients within each group are very high (>0.8). A high number of correlated features makes it difficult for the black-box optimizer to find the best parameters and the parameters are quite unstable.

The ϵ-summand added when computing the value is sampled from a Gaussian distribution with zero mean and unit variance. It can be interpreted as the usefulness measure of a random sampling agent. We found that adding it increased the performance of an active learning agent with this policy significantly for some tasks. We assume that it prevents the active learning agent from being stuck in local minima, just like the stochastic component of stochastic gradient decent increases the performance of the back-propagation algorithm when training neural networks.

The beta-parameters could be chosen such that the agent performs exactly like one single heuristic, e.g. by setting all betas to zero, the agent would perform pure random sampling, or by setting β_1 to a high value and the other betas to zero, the agent would perform pure uncertainty sampling. Thus, assuming this agent learned the best β-parameters for a task, its worst-case performance is the one of the best heuristic on this task, but it can also perform significantly better.

3.4.2. Choice of parameters of linear policy

The linear policy is a policy parametrised by the β-parameters. We decided to learn which parameters are good using Monte Carlo policy search. Different to the first approach, the objective is not to predict the improvement of the supervised learning model when choosing a sample to be labelled (i.e. a regression problem), but rather to maximize the performance of the supervised learning model after a complete episode. As it is very hard to attribute the performance at the end of an episode to a specific action or, even harder, to a specific choice of beta, we treat the optimization problem as a black-box optimization problem. The objective function is defined in Algorithm 7 and calculates the performance on the choice of β.

Algorithm 7 Pseudocode for objective function of Monte Carlo policy search

```
Require: trainingTask, activeLearningEnvironment, MonteCarloAgent
  function objectiveFunction(β)
      agent ← MonteCarloAgent(β)
      for i ← 1 to noEpisodes do
          results ← activeLearningEnvironment.run(trainingTask, agent)
          performances[i] ← results.getFinalPerformance()
      end for
      meanPerformance ← mean(performances)
      return meanPerformance
  end function
```

Taking the mean of several active learning runs reduces the noise of the objective function. We set the number of episodes such that all runs could be run in parallel on the server used, which equalled 104 episodes for the tasks using a random forest and 26 episodes for the tasks using a deep neural network, which already uses multi-threading internally. We found the maximum of the objective function using tree parzen estimators [60], as they are very sample-efficient and search for the global optimum.

This approach allows to combine the research results on heuristics and learning AL approaches and keeps their advantages: Like heuristics, its policy is easy to understand, and thus its behaviour can be predicted quite good. Nonetheless, it has the adaptability of learning AL frameworks and allows to combine different features and heuristics. Its main disadvantage are the computational costs: one evaluation of the black-box objective function includes several complete active learning runs, and thus multiple trainings of a supervised learning model. Furthermore, the greedy sequential choice of the next sample to be added to the batch is non-optimal, as already pointed out by Kirsch et al. [48].

3.5. Approach 3: Uncertainty-weighted clustering

Both the Q-Learning and the heuristic ensemble approach have the disadvantage, that they choose the samples sequentially in a greedy manner, which is non-optimal. An approach to choose all samples at once is the core-set selection approach by Sener and Savarese [14]. They showed that the generalization error is upper-bounded by the maximum of minimum distances of the unlabelled samples to a labelled one. Minimizing this upper bound equals solving the k-Center problem [61]. As the problem is NP-hard they solve it using a greedy algorithm.

If one relaxes the k-Center problem such that not the maximum distance to the nearest center, but the sum of distances to the nearest center should be minimized, it becomes the k-median problem. They tried an active learning agent choosing the cluster centers of the k-median as samples to be labelled, but it performed quite badly. They assume the reason is that k-median clustering ignores the set of already labelled samples, and thus may choose samples very similar to them, while their k-center approach does not cover these spaces twice.

We propose to relax the k-Center problem in two ways: First it is allowed to choose artificial center points (not necessarily unlabelled points) . Second, not the maximum of the distances of the unlabelled samples to the nearest center points should be minimized, but rather the sum of the squared distances to the nearest center point. Using this relaxation, the k-center problem becomes a k-means clustering problem [62], for which very efficient non-greedy algorithms exist. One very important difference to the clustering agent tried by Sever and Savarese [14] and the classical k-means problem is, that we assume that there are already some fixed centroids, which represent the already labelled points.

3.5.1. Solving k-means problem with some fixed centroids and weighted points

We use and adapted the expectation-maximization algorithm to solve the k-means problem [63] with some fixed centroids. We further added the option to assign different weights to the

unlabelled points to be clustered. After the convergence of the k-means algorithm, existing unlabelled points have to be chosen, while the centroids can be at any point. Thus, we replace each centroid by the closest unlabelled point to it. The pseudocode for this adapted k-means algorithm is given in Algorithm 8.

Algorithm 8 Pseudocode of the k-means algorithm with some fixed centroids and weighted points

Require: $n_{centroids}, D_{unlabelled}, D_{labelled}, weights$
  $centroids \leftarrow initializeCentroids(n_{centroids}, D_{unlabelled})$
  while *not converged* **do**
    $allCentroids \leftarrow concat(centroids, D_{labelled})$
    $closestCentroidIndices \leftarrow pairwise\_distances\_argmin(D_{unlabelled}, allCentroids)$
    for $i \leftarrow 1$ *to* $n_{centroids}$ **do**
      $pointsBelongingToCluster \leftarrow D_{unlabelled}.where(closestCentroidIndices == i)$
      $weightsBelongingToCluster \leftarrow weights.where(closestCentroidIndices == i)$
      $centroids[i] \leftarrow weightedMean(pointsBelongingToCluster, weightsBelongingToCluster)$
    end for
  end while
  $centroids \leftarrow closestPoints(centroids, D_{unlabelled})$
  return *centroids*

One problem with the fixed centroids is that they act like a border preventing the unlabelled points from getting behind them. They cause the problem to have many local minima. We found that this problem becomes much smaller when using a better initialization than random initialization. We use an adapted version of the k-means++ initialization [64]: First it is assumed that the points in the labelled set are set as initial centroids. Then the next point chosen as initial centroid is the one having the highest utility, which is defined as the product of the minimum distance of the point to the already chosen centroids and its weight. It is added to the already chosen centroids. This algorithm, whose pseudocode is given in Algorithm 9, differs from the classical k-means++ algorithm in two ways: First, we weight the points, thus points with higher weights have a higher chance to be chosen. Second, the classical algorithms chooses the next point with a probability proportional to its squared minimum distance. We tried this approach, but it was computationally unstable, thus we switched to always choosing the point with the highest utility deterministically.

Algorithm 9 Pseudocode for the k-means++ initialization with some fixed centroids and weighted points

Require: $n_{centroids}, D_{unlabelled}, D_{labelled}, weights$
  $centroids \leftarrow D_{labelled}$
  for $i \leftarrow 1$ *to* $n_{centroids}$ **do**
    $minimumDistancesToCentroids \leftarrow pairwise\_distances\_min(D_{unlabelled}, centroids)$
    $pointUtilities \leftarrow minimumDistancesToCentroids * weights$
    $bestPointIndex \leftarrow argmax(pointUtilities)$
    $bestPoint \leftarrow D_{unlabelled}[bestPointIndex]$
    $centroids \leftarrow concat(centroids, bestPoint)$
  end for
  $centroids \leftarrow centroids.remove(D_{labelled})$
  return $centroids$

3.5.2. Learning entropy-based weighting of points

As already found in the former approaches, the utility of choosing a sample may depend the entropy of its prediction. Thus, we decided to allow the agent to weight points differently based on their entropy according to Equation 3.5. The weights are an exponential function of the prediction entropies, with an entropy weight factor controlling the sign and amount of the dependency: A high entropy weight factor causes the weights of uncertain samples to be much higher, causing the agent to behave more like an uncertainty sampling agent. An entropy weight factor of zero weights all samples the same, a negative weight assigns a higher weight to samples with a lower entropy.
We found that the best entropy weight factor depends not only on the dataset, but also on the current state of the active learning process. Mostly a lower entropy weight was preferred in the beginning, when the model is still quite inaccurate, and thus the decision border is not close to the decision border in the end. To allow a dynamic entropy weight, two entropy weight factors dependent on the current relative use of the annotation budget are added. To decorrelate the base entropy weight factor from them, they are defined such that the entropy weight factor equals the base entropy weight factor if exactly half of the annotation budget is used.

$$\begin{aligned} weights &= exp(entropyWeightFactor * predictionEntropies) \\ &\quad with \\ entropyWeightFactor &= entropyWeightFactor\_base \\ &\quad + entropyWeightFactor\_relBudget * (relBudget - 0.5) \\ &\quad + entropyWeightFactor\_relBudget\_sqrt * (\sqrt{relBudget} - \sqrt{0.5}). \end{aligned} \tag{3.5}$$

The three entropy weight factors are treated like the beta-parameters of the approach using Monte Carlo policy search. The objective function is again to maximize the accuracy at the end of an episode as defined in Algorithm 7. The black-box algorithm used to solve this problem are again adaptive tree parzen estimators [60].

Just like the this Monte Carlo agent, this agent can be easily extended to use more features, e.g. prediction variance measures using BALD, making it quite flexible. The k-means algorithm does not allow a trade-off between diversity sampling and representativeness sampling, while the approach with Monte Carlo policy search allows it, which might be a disadvantage in datasets containing many outliers or being very unbalanced.

4. Experiments and Results

The first sections of this chapter outline the experimental setup: The tasks on which the agents were evaluated are explained, followed by a description of the heuristic agents, which are used as benchmarks. Next the procedures for training and evaluating the agent with the corresponding parameters are described.
The experiments have shown that there is a lot of variance or randomness in many parts of the training and evaluation procedures, which made it necessary to perform experiments many times. This problem and other problems of training and evaluating the learning agents are described in another section.
The last section shows the results of the heuristics and learning agents on the different evaluation tasks.

4.1. Tasks and datasets

In total 20 different datasets with three different supervised learning models were used:

- Gaussian clouds dataset with a random forest
- checkerboard datasets (2x2, rotated 2x2, 4x4) with a random forest
- 11 datasets from the UCI repository (0-adult, 1-australian, 2-breast cancer, 3-diabetes, 4-flare solar, 5-heart, 6-german, 7-mushrooms, 8-waveform, 9-wdbc, 10-spam) with a random forest
- MNIST, fashion-MNIST and CIFAR10 datasets with a CNN
- bAbI single supporting fact and two supporting facts (Question answering) with a LSTM

Their sources and properties are described in the following subsections.

4.1.1. Binary classification tasks on non-structured data

The first three groups of datasets with a random forest as classifier are all binary classification tasks with non-structured data.

The Gaussian clouds task generates an artificial dataset on-the-fly by defining two two-dimensional Gaussian clouds by their parameters. It draws 400 samples from them and then sets the objective to determine whether a sample was drawn from one cloud or the other. First the proportion of class zero to class one is determined by sampling from a uniform distribution in [0.1; 0.9]. Then both the mean and the covariance matrix of each cloud are again sampled randomly. An example for a such a generated dataset is shown in Fig. 4.1.

All checkerboard tasks are XOR-like binary classification tasks with two-dimensional input data. The samples are pre-computed and not generated randomly like for the Gaussian clouds task. The full datasets are shown in Fig. 4.1.

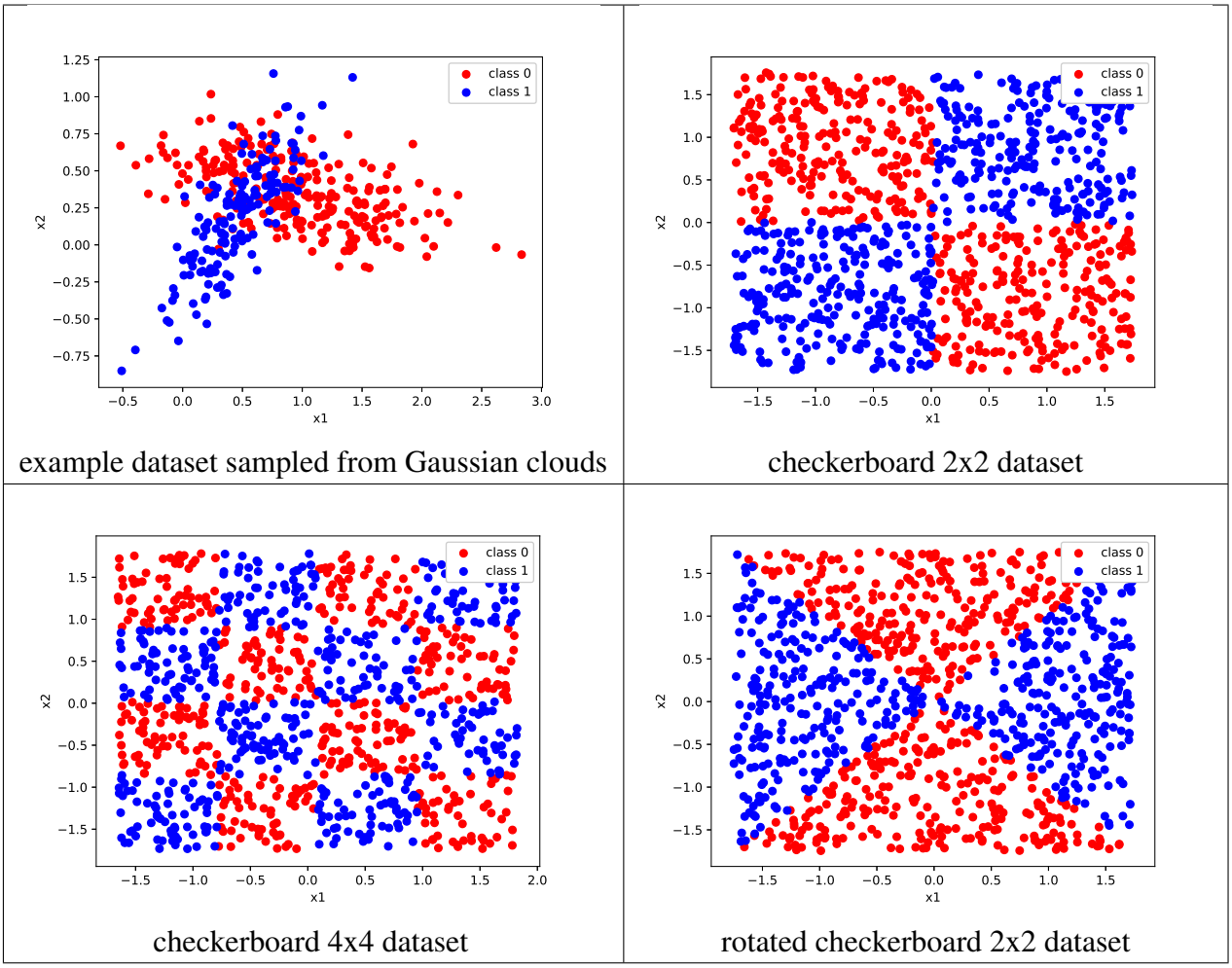

Figure 4.1.: Binary classification tasks on 2d-data.

The UCI tasks use datasets from the UCI Machine Learning repository [65]. They are also binary classification tasks, but have a mix of categorical, integer and continuous attributes.

These tasks were chosen because they were already used by Konyushkova et al. [6], [59] in their papers about learning active learning. This allowed us to estimate which results are realistic and achievable and it allowed to re-use parts of the code for generating the datasets.

To facilitate the comparison with the results by Konyushkova et al. [6], the same classifier they used was used: a random forest implemented using the scikit-learn library [66].

4.1.2. Image classification tasks

The three image classification tasks for evaluating the agent are using the MNIST [67], fashion-MNIST [68] and CIFAR10 [69] dataset and a three-layer convolutional neural network implemented in keras [70] as classifier. The MNIST and fashion-MNIST dataset both

contatin grayscale 28x28 pixel images, and have 60,000 training samples and 10,000 training samples. The CIFAR10 dataset shows objects like vehicles and animals and also has 10 mutually exclusive classes. Its images have 32x32 pixels and are coloured. It has 50,000 training images and 10,000 test images.
Because Prateek et al. [32] have shown that performance differences between different active learning agents may vanish when more regularization techniques are used, we tried to maximize the performance of the random baseline using following techniques: random data augmentation, dropout, L1- and L2 regularization and hyperparameter optimization using tree parzen estimators [60]. The hyperparameters optimized are the dropout rate, the learning rate, the number of neurons in each layer, the L1- and L2-regularization factor and the number of epochs. They were chosen such that the categorical cross-entropy loss when training on 200 random samples is minimized. The best parameter combination for the MNIST dataset was also used for the fashion-MNIST and CIFAR10 dataset. We did not use more advanced networks like the ResNet or VGG16 because they are computationally much more expensive and tend to overfit their huge number of parameters on the very small training set size.

4.1.3. Question answering tasks

The bAbI dataset [71] is a collection of question answering datasets of which two different challenges are used: one with a single supporting fact and one with two supporting facts. The classifier used is using a long short term memory and taken from a keras example, which follows a paper by Sukhbaatar et al. [72]. Like for the image classification tasks, the hyperparameters were optimized to minimize the loss on a small random subset.

4.2. Alternative agents as benchmarks

Additional to the learning agents, several more agents were developed to provide benchmarks to compare the learning agent against. These are the following:

- Random sampling:
  This active learning agent is the simplest one: It chooses the sample to add to the batch randomly, thus its performance does not differ between batch-mode and sequential active learning. Even though it is so simple it still outperforms heuristics on some datasets.
- Uncertainty sampling:
  This active learning agent is both very commonly used and easy to implement. We use the entropy-based uncertainty sampling, which chooses as next sample to add to the batch always the one having the highest entropy of the prediction vector.
- Bayesian active learning by disagreement (BALD):
  This agent calculates the uncertainty of the prediction using Monte Carlo dropout, as described before in item 2.3.1. We found that it performed on par or slightly worse than uncertainty sampling, thus we do not include it in our plots.

- Algorithmic expert:
  The algorithmic expert is not a true active learning agent, as it knows the true labels of all samples in the unlabelled set. It can calculate the improvement of the supervised learning model if a sample is added to the labelled set by adding this sample and re-training the classifier. Then it chooses the sample with the highest improvement to be labelled next. This way it provides an upper bound for the maximum performance achievable by a perfect active learning agent. By design, the algorithmic expert works as a sequential agent.
  As computing the improvement is quite expensive, we only compute it for a random subset of 10 samples out of the labelled set. Even though this decreases the performance of the algorithmic expert, it still provides a very strong benchmark and easily outperforms heuristics.

4.3. Setup and parameters

Before applying a learning agent, it has to be trained on one or many training tasks. As already found out by Konyushkova et al. [59], training on multiple datasets increases the performance of learning active learning strategies.

The procedure for training the Q-learning agent was already described in subsection 3.3.3. We only trained the agent on the Gaussian clouds task, as the procedure is computationally extremely costly and this is the computationally cheapest task.

The experiment procedure for training the approach using Monte Carlo policy search is the following: First the β-parameters maximizing the objective function defined in Algorithm 7 are found by using the black box optimizer of adaptive tree parzen estimators. The prior for the β-parameters was set to be a normally distributed random variable with fixed location and variance, thus the agent was allowed to learn both a negative an positive effect of a certain feature on the performance.
The training was performed iteratively by using the best β-parameters found at one task as location priors for training on the same task again or another task. Once the β-parameters were found to be sufficiently good, usually after a few hours of optimizing, these parameters were saved in the learning agent. Then this learning agent was applied on the same task and/or similar evaluation tasks.

The three uncertainty weight parameters of the clustering agent are learned the same way using the same objective function, black-box optimizer and training procedure.

The size of the starting set was chosen to be 8 for the binary classification tasks and 40 for the classification tasks with 10 classes, thus the expected number of samples per class is 4. The starting set size was set to 40 for the question answering tasks. It is sampled randomly. The annotation budget was chosen such that the submodularity of active learning is clearly visible as the curves begin to become flat. The batch size was set dependent on the annotation budget such that about 10 batches are filled till the annotation budget is exhausted.

Not all heuristic benchmarks are used on all tasks: As the algorithmic expert is computationally very expensive, it is only used on the binary classification tasks. Sequential uncertainty sampling is used for all tasks, batch-mode uncertainty sampling is only used if sequential uncertainty sampling performs better than random sampling. The performance of random

sampling is independent of the batch size, thus batch-mode random sampling is used to save computation effort. The number of repetitions per agent were chosen such that the differences between agents are significant, which needed between 13 repetitions for uncertainty sampling applied on the vision tasks and 1000 repetitions for uncertainty sampling on the UCI tasks.

4.4. Problems during experiments

The biggest problem during all experiments was the very high variance of all parts of it. The sources for these randomness are the following:

- Randomness in the active learning task:
  The Markov decision process has randomness in many parts: The first one is the choice of the labelled starting set, which is a randomly sampled small subset of the total training set. Second, the training of a supervised learning model also includes stochastic processes: For the random forest it happens in the bootstrapping, for neural networks it occurs in the choice of the initial weights, the split of the training data into batches as part of stochastic gradient descent and inside the dropout layers.
- Randomness in the policy of an active learning agent:
  The active learning agent might have random components in its policy. The Monte Carlo Agent has randomness as the ε-summand for calculating the usefulness of a sample is normally distributed. The Q-Learning agent is deterministic during evaluation time, but as the training is performed with an ε-greedy policy, thus it includes both the randomness whether the Q-learning agent or random agent is used and the randomness of the random agent. Only uncertainty sampling and the uncertainty-weighted clustering agent are deterministic.
- Randomness in training an active learning agent:
  The training of an active learning agent includes the randomness in the active learning task and the one in the policy of the agent. It additionally has the randomness in training a reinforcement learning agent. As the Monte Carlo agent and the uncertainty-weighted clustering agent use black-box optimization, they include the randomness of the initial choice of parameters to evaluate and the one included by the adaptive tree parzen estimators.
  The Q-learning agent's supervised learning model for fitting the Q-function is a random forest, thus it uses the stochastic process of bootstrapping. Furthermore, the improvement of the loss (i.e. the reward of the MDP) is computed for 10 samples in every step, which are sampled randomly.

This high randomness made it necessary to repeat runs many times before any significant results could be achieved.

Both the Monte Carlo and the uncertainty-weighted clustering agent worked quite out-of-the box with little need to manually tune their hyperparameters and design choices. The main problem of the Monte Carlo agent was the high correlation of some of its features, which led to instability of the found parameters. This problem was reduced by reducing the number of features to only six, it might be possible to use even less features. Based on this knowledge,

the features of the uncertainty-weighted clustering agent were design to be uncorrelated, thus it did not suffer from this problem.

Contrary to these two approaches, the Q-Learning has shown to have many and more severe problems:

- Very high randomness of targets of training data: The target of the Q-learning function, the improvement when adding a sample to the labelled set, is very noisy. The main reason is that the variance of the accuracy of the supervised learning model is very high compared to the improvement when adding a specific sample, thus the probability that the improvement is positive is only slightly higher than 50%. This very high variance of the targets makes it hard to fit the Q-function. Both the Monte Carlo and uncertainty-weighted clustering agent do not suffer from this problem, as their objective is the accuracy of the final performance of several runs. Both taking the final accuracy, which is the sum of all improvements, and the mean of several runs reduces the variance of the objective drastically, thus they suffer much less from the randomness than the Q-learning agent does.
- Greedy behaviour:
  The use of a discount factor of 0, thus only maximizing the reward in the next step, fosters greedy behaviour. Using non-zero discount factors would be possible, but bootstrapping would make the training even more complicated and finding a good discount factor is hard.
- Very high computational cost:
  Generating one training sample for the Q-learning agent includes the training of one supervised learning model. Even using the Gaussian clouds tasks where a random forest was trained on 2-dimensional data with only between 8 and 40 data points, generating 100,000 training samples using the algorithmic expert took a few hundred CPU core hours. Generating training data using a CNN as classifier, coloured image data and a few hundred samples takes orders of magnitudes longer, making the Q-learning approach very unpractical.
- High storage need for policy:
  We found that the Q-learning agent performed best, when the random forest used for estimating its Q-function had a few thousand estimators and a high depth, which led to random forests needing multiple Gigabytes of space to be stored. When training multiple agents differing in hyperparameters, training tasks and batch sizes, this quickly led to problems with memory.
- Low understandability:
  It is very difficult to understand what the Q-learning agent learned. This made it difficult to find out, why an agent performed good or bad and how it could be improved.
- No reproducibility:
  While we found Q-learning agents, that performed well on a certain task, we could not reproduce them. Generating the training data again, even with the same parameters, resulted in agents with completely different performance. This is due to the different random seeds used in each run.

These problems are not unexpected, but rather common in reinforcement learning settings, as summed up by Irpan [73]. Because we could not reproducibly train a Q-learning agent and it

was very costly to train it even on very simple datasets, we decided to abandon this approach and do not report any results for it.

4.5. Results

This section shows the performance of the active learning agent on the tasks as plots of the classification accuracy over the number of labelled samples. The plots for the different evaluation tasks all share the same structure: The dark shaded area around the curves of each agent report the 95% confidence intervals, the light shaded areas are the standard deviations. A suffix to the agent names in the form of '_batchSize' denotes the batch size the agent used or whether it was applied sequentially. There is no such suffix for the algorithmic expert, always performing sequentially and random sampling, performing independently of the batch size.

4.5.1. Results on UCI datasets

The UCI tasks 2-breast_cancer through 9-wdbc were used as training tasks to train the learning agents, the UCI tasks 0-adult, 1-australian and 10-spam were used as evaluation tasks. The training of the agents was performed with a starting size of 8, batch size of 4 and annotation budget of 40. The results for the three evaluation tasks are shown in three figures: 4.2, 4.3 and 4.4. The relative performance of the agents is quite similar across the three datasets: The algorithmic expert performs very well on all tasks, outperforming all other agents by far. Uncertainty sampling performs well with sequential uncertainty sampling performing slightly better than batch-mode uncertainty sampling. The performance of the Monte Carlo agent lies between the two uncertainty sampling heuristics. The uncertainty-weighted clustering agent does not perform as well as these agents, but at least better than random sampling. One should note that the standard deviations of the agent's accuracy can reach up to 0.1, which is far more than the difference between the agents, thus the final accuracy of the model depends much more on the random seed than on the active learning agent chosen.

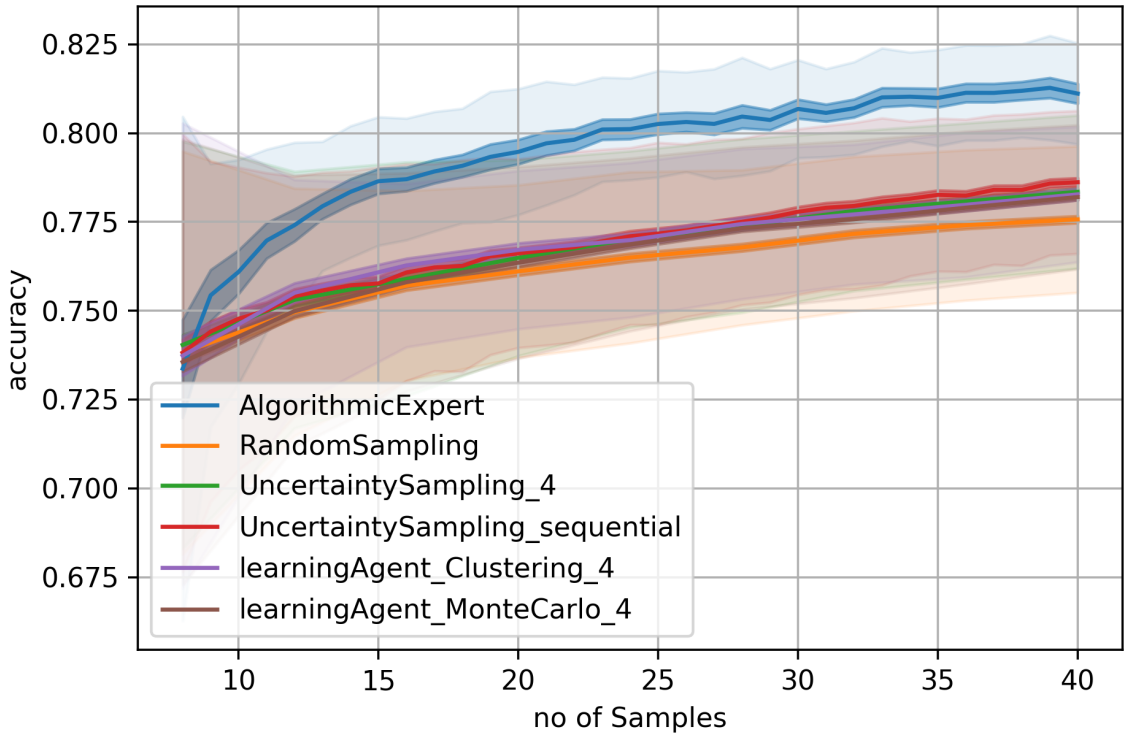

Figure 4.2.: Results on 0-adult dataset: The learning agents perform on par with uncertainty sampling and better than random sampling.

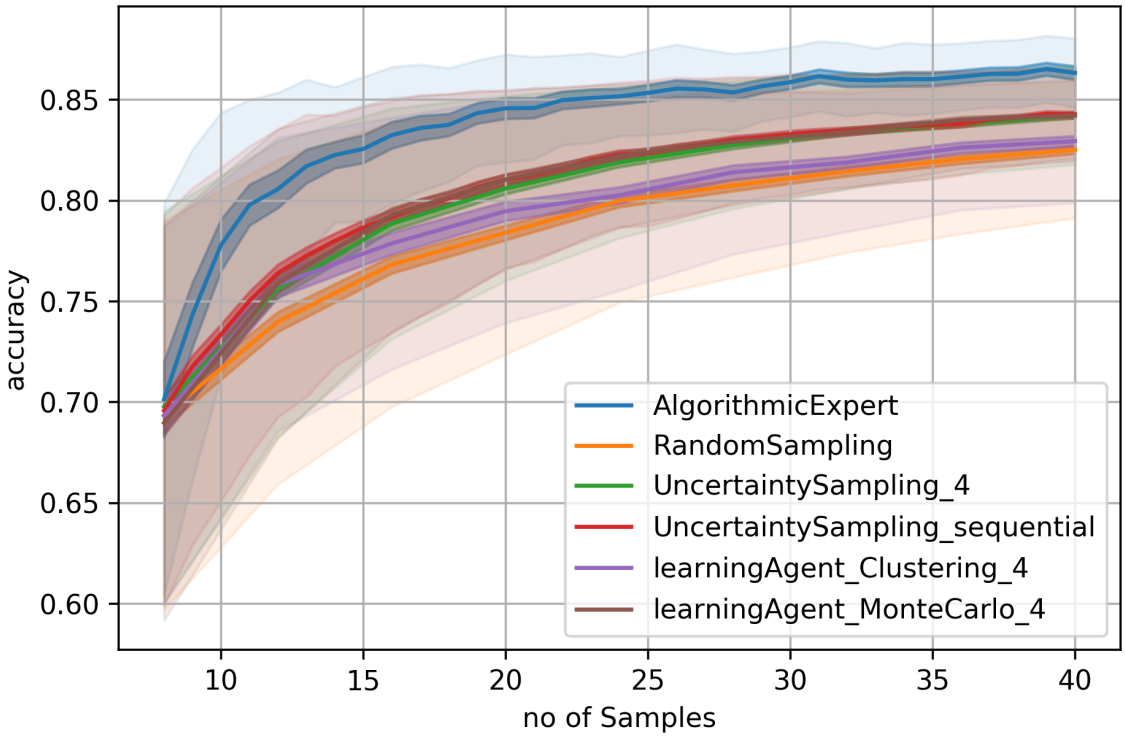

Figure 4.3.: Results on 1-australian dataset: The Monte Carlo agent performs on par with uncertainty sampling and better than random sampling und the uncertainty-weighted clustering agent.

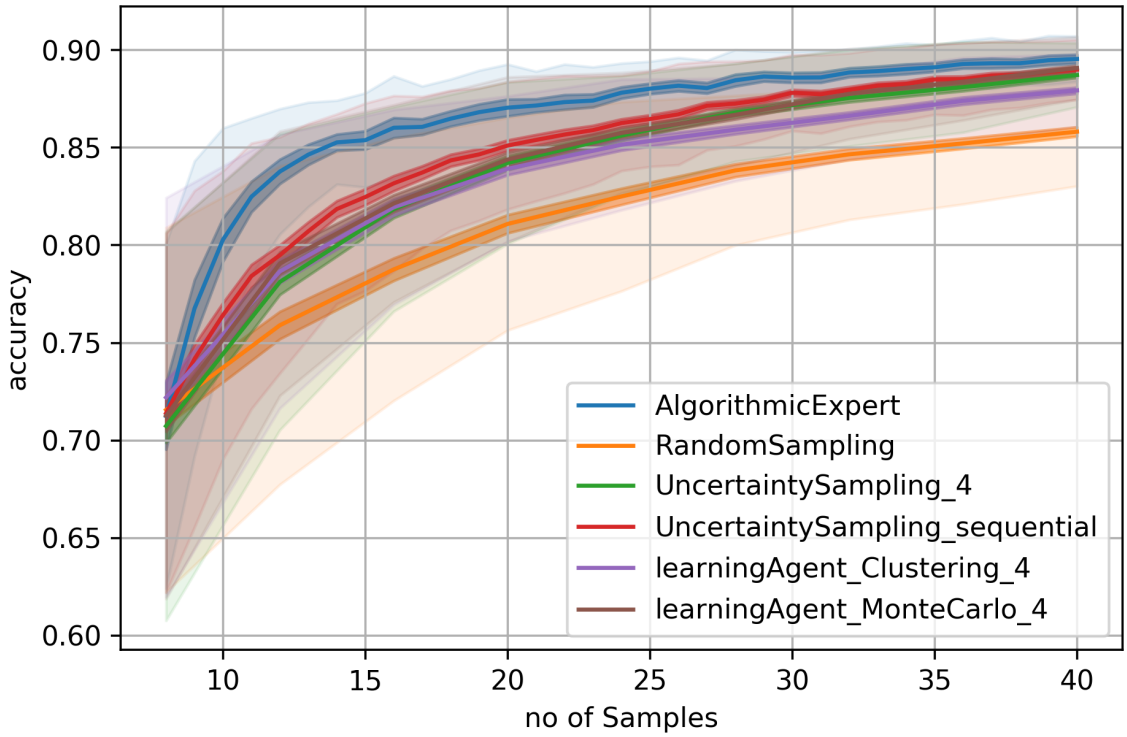

Figure 4.4.: Results on 10-spam dataset: The Monte Carlo agent performs on par with uncertainty sampling and better than random sampling und the uncertainty-weighted clustering agent.

4.5.2. Results on checkerboard datasets

The checkerboard tasks checkerboard 2x2 and rotated checkerboard 2x2 were used as training tasks to train the learning agents, the checkerbard 4x4 task was used as evaluation task. The training of the agents was performed with a starting size of 8, a batch size of 32 and an annotation budget of 72.

The plot 4.5 shows again, that the algorithmic expert outperforms all heuristics by far. Sequential uncertainty sampling performs quite good in the beginning, but performs worse than random sampling in the end. Both learning agents outperform the algorithmic expert after about 120 samples. They reach this impressive performance even though they always choose a batch of 32 samples at once. We also trained and applied these two learning agents with a batch size of 4, the performance did not differ significantly. One should note that the checkerboard 2x2 and rotated checkerboard 2x2 tasks are much simpler than the checkerboard 4x4 task, the performance of the learning agents might be even better if they were trained on a harder and more similar task like the checkerboard 3x3 or 3x5 task.

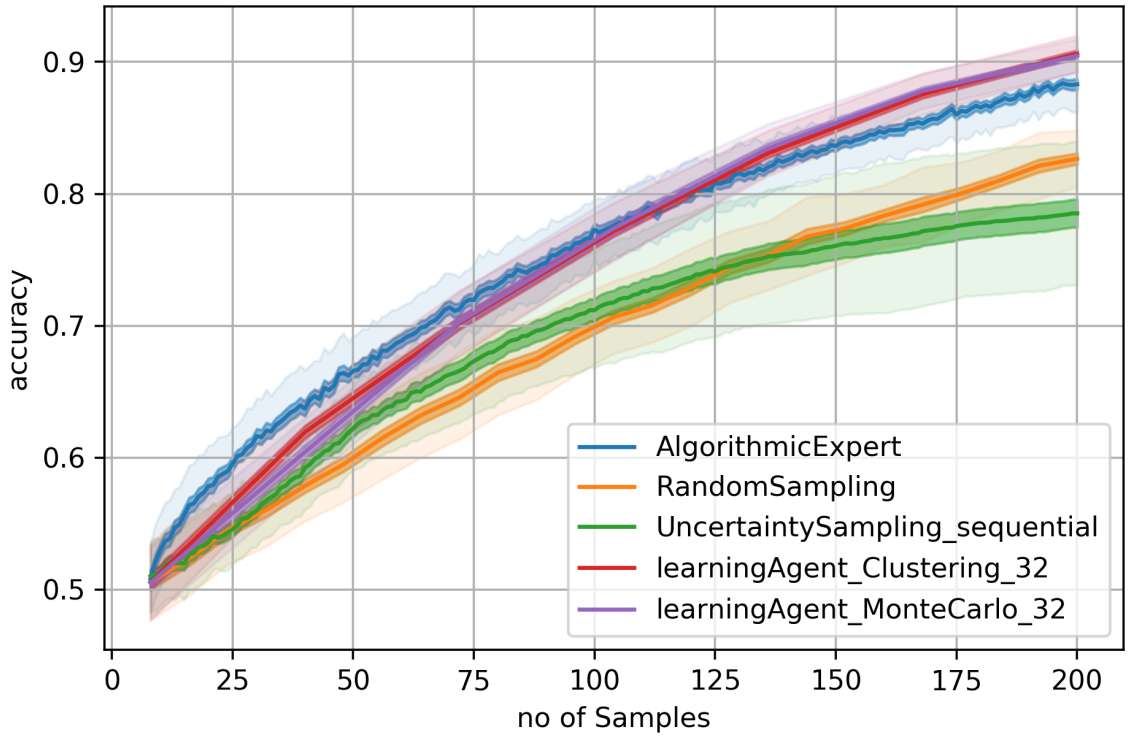

Figure 4.5.: Results on checkerboard 4x4 task: The learning agents peform very strongly and even outperform the algorithmic expert.

The algorithmic expert in this example tried 10 random samples per step over 192 steps per episode and 104 episodes, which equals about 200.000 trainings of a random forest classifier to generate its curve. This is still feasible for the computationally cheap training of small random forest on 2-dimensional data, but not anymore for more expensive classifiers like deep neural networks with more parameters and applied on much larger input data.

4.5.3. Results on (fashion-) MNIST tasks

The MNIST task was used as training task to train the learning agents, both the MNIST and fashion-MNIST task were used as evaluation tasks. Training and evaluating on the same task is not possible in practical settings, but it gives an intuition for the maximum performance reachable if the training tasks are very similar to the evaluation task. The agents were trained using a starting size of 40, a batch size of 32 and an annotation budget of 168, even though they were evaluated with an annotation budget of 360. The reason is that an episode with a smaller annotation budget takes much less computation time, allowing to perform more evaluations of the objective function in the same time.

On the MNIST task shown in Fig. 4.6, sequential uncertainty sampling performs approximately on par with random sampling. It has a performance drop every 16 samples, we assume this is due to the batch size of training the CNN of 16. Both the Monte Carlo and clustering agent outperform both heuristics significantly. We also tried the Monte Carlo agent with a batch size of 4 and 64, but the results were the same.

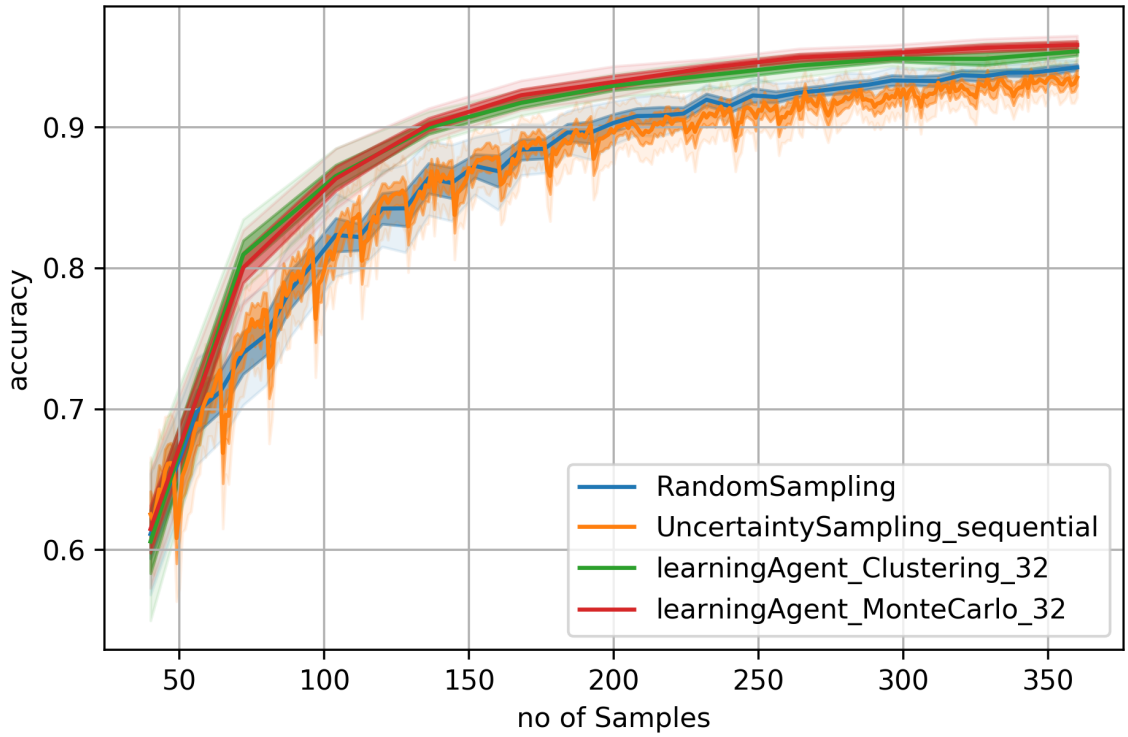

Figure 4.6.: Results on fashion MNIST dataset: Both learning agents outperform the heuristics.

On the fashion-MNIST task shown in Fig. 4.7 both learning agents still outperform random sampling, but by a much smaller margin. As both learning agents rely partly on uncertainty sampling, the reason might be that uncertainty sampling works quite badly on this task.

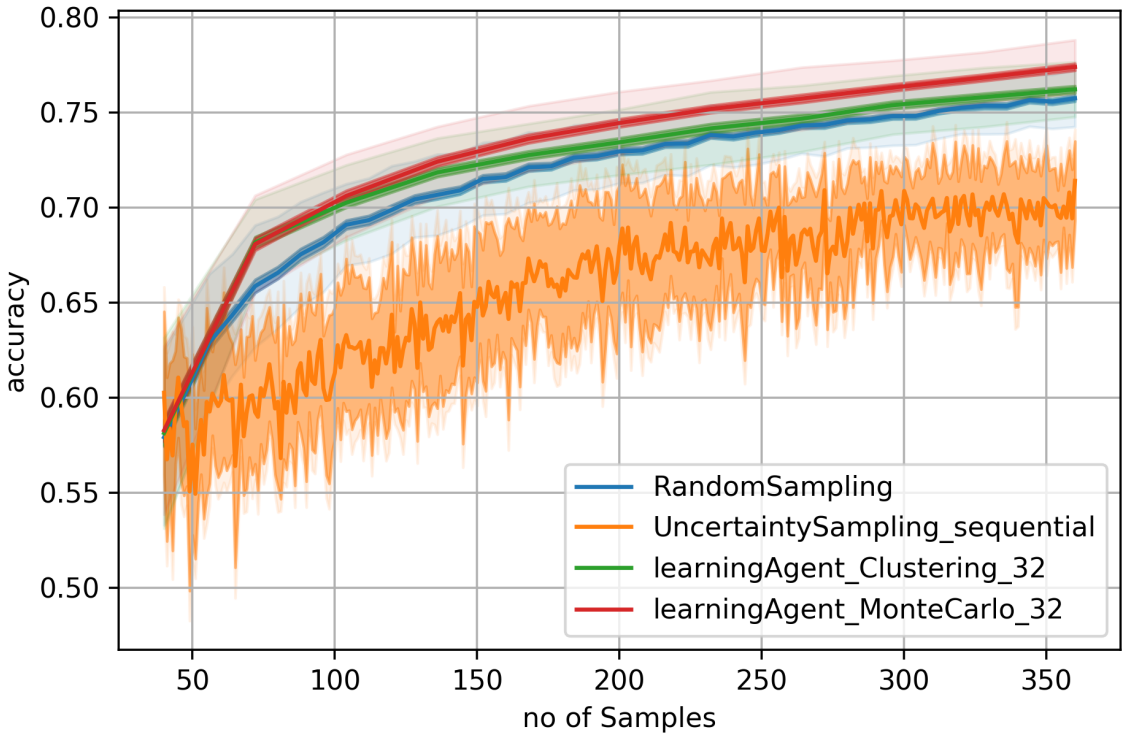

Figure 4.7.: Results on fashion MNIST dataset: Both learning agents outperform the heuristics.

4.5.4. Results on CIFAR 10

We found that the agent trained on MNIST did not perform well on CIFAR10, thus we tried how good an agent trained on CIFAR10 performs on CIFAR10. We assume, that the MNIST task is too simple to be a good training task for CIFAR10. Just like for the (fashion-) MNIST tasks, the starting size was 40, the batch size 32, the annotaton budget when training was 168. The learning agents still outperform random sampling by a very small, but still significant, margin as shown in Fig. 4.8.

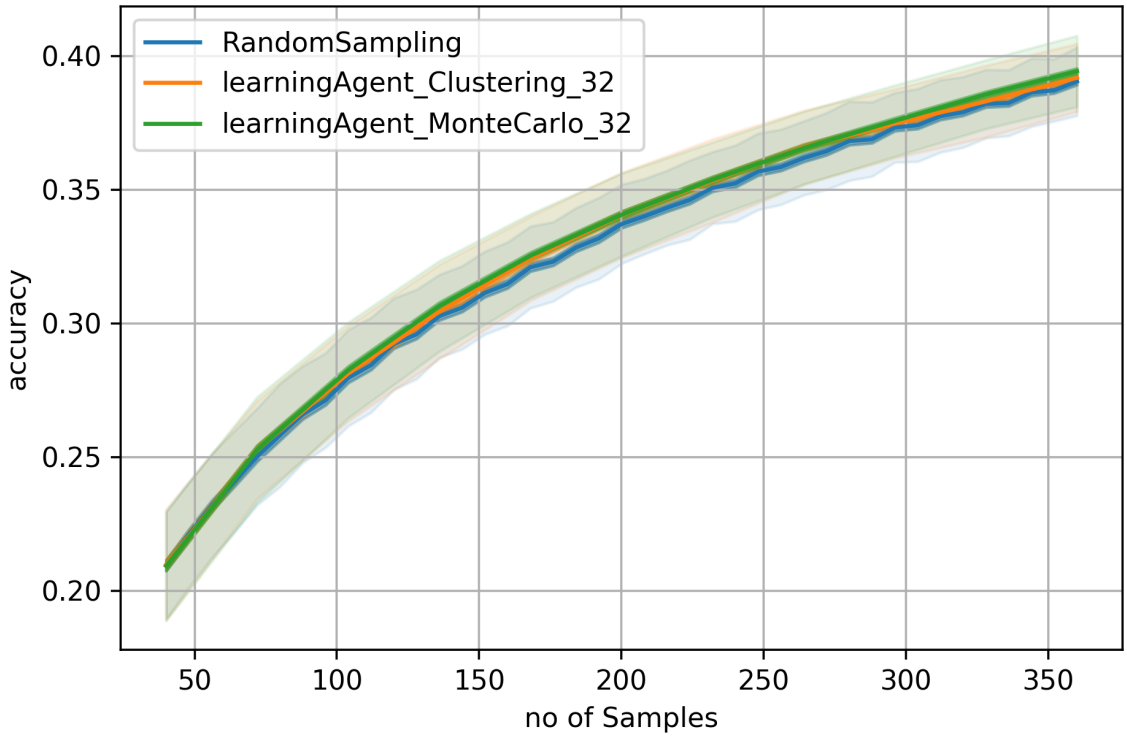

Figure 4.8.: Results on CIFAR10 dataset: Both learning agents outperform random sampling, but only by a very small margin.

4.5.5. Results on question answering tasks

The bAbI task with a single supporting fact was used as training task to train the learning agents, it is also used as evaluation task with the results shown in Fig. 4.9. Additionally the bAbI task with two supporting facts is used as evaluation task, the results are shown in Fig. 4.10. Just like for MNIST and CIFAR 10, the agents were trained using a starting size of 40, a batch size of 32 and an annotation budget of 168. Both random sampling and uncertainty sampling perform poorly on these tasks and the clustering learning agent is only slightly better. The Monte Carlo agent performs much better than all the other agents.

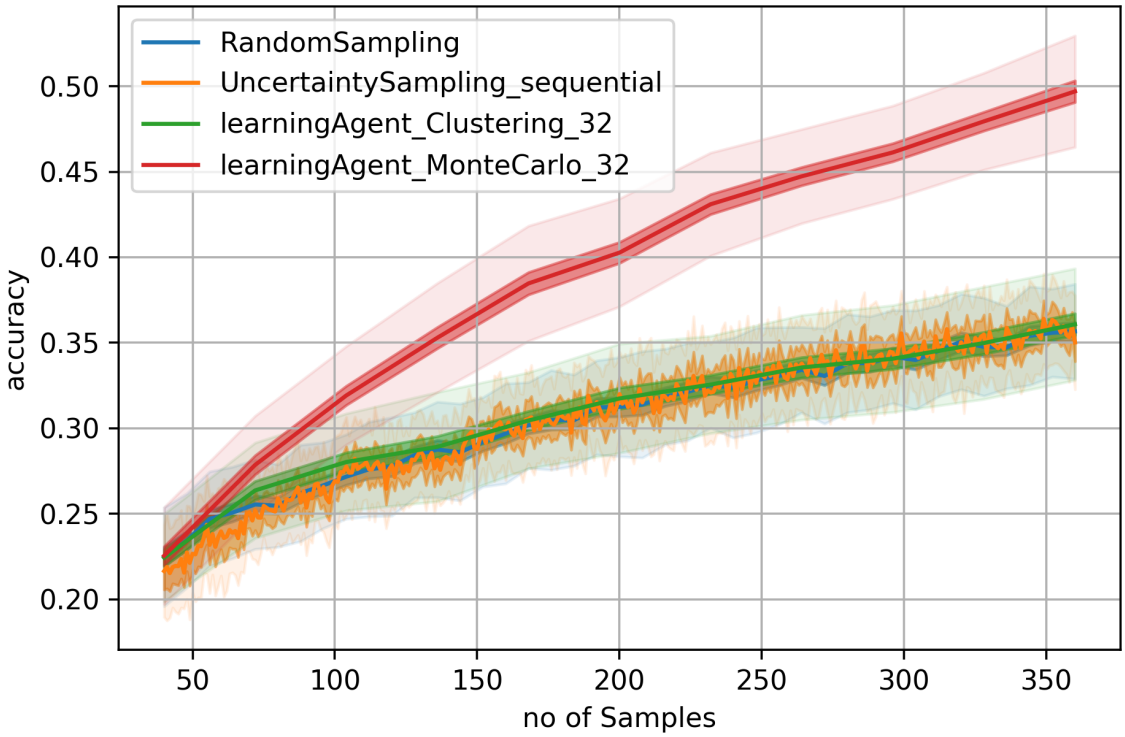

Figure 4.9.: Results on QA task with bAbI dataset single supporting fact: The Monte Carlo agent outperforms the other agents by far.

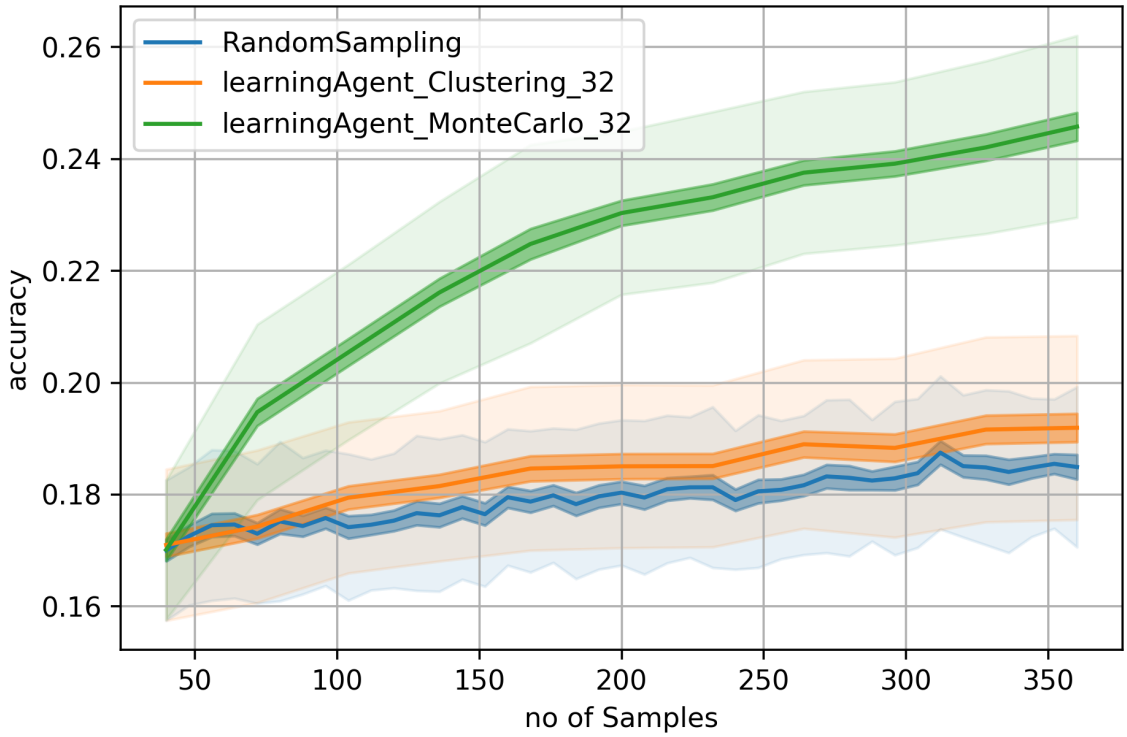

Figure 4.10.: Results on QA task with bAbI dataset two supporting facts: The Monte Carlo agent outperforms the other agents by far.

5. Analysis and discussion

This chapter aims to understand the learning agents and their policy better. The first section compares the learning agents with heuristics on a theoretical basis, the second one shows their sample choice using an example task. It allows to get an intuition about the policy of the learning agents and their advantages and disadvantages. The third and fourth section analyse the parameters learned on the training tasks by the Monte Carlo and uncertainty-weighted clustering agent respectively. This allows to understand which features are important in general and for specific tasks and how the importance of features is related to properties of the dataset. This chapter finishes with an outline of promising future research directions.

5.1. Theoretical comparison of proposed active learning approaches and heuristics

In Table 5.1 we compare heuristic active learning approaches and the three approaches proposed in this thesis with each other concerning their understandability, robustness, adaptability, combination of different features, computational cost and other metrics. It shows that the approaches using Monte Carlo policy search and uncertainty-weighted clustering are able to combine the advantages of heuristics and Q-learning, while only sharing part of their disadvantages.

Table 5.1.: Summary of advantages and disadvantages of learning active learning vs. heuristics

Approach	Advantages	Disadvantages
Heuristics	• policy is easy to understand • known, predictable behaviour	• rarely combining several features and ideas • no best heuristics for all datasets and tasks • some heuristics select the samples greedily
Approach 1: Q-Learning	• adaptive to dataset, supervised learning model and metric to optimize • combines several features in non-engineered way	• computationally very expensive to train • hard to implement • discards research on heuristics • greedy selection of samples • problems of reinforcement learning: credit assignment problem, sensitivity to hyperparameters and random seeds, many local minima
Approach 2: Monte Carlo policy search	• adaptive to dataset, supervised learning model and metric to optimize • combines several features in non-engineered way • policy is easy to understand	• computationally expensive to train • greedy selection of samples
Approach 3: uncertainty-weighted clustering	• adaptive to dataset, supervised learning model and metric to optimize • combines several features • policy is easy to understand	• computationally expensive to train • partly engineered solution with only limited adaptiveness, e.g. no trade-off between diversity sampling and representative sampling possible

5.2. Comparison of proposed active learning approaches and heuristics on example task

To get an intuition for the proposed agents, consider again the example task shown in Fig. 2.2 with one Gaussian cloud for each class located at (0,0) and (1,1) respectively. A random forest was trained on this classification task with a total of 1000 samples per class and 8 random samples in the labelled set. The following plots show how an uncertainty sampling agent, an agent using Monte Carlo policy search and an agent using uncertainty-weighted clustering choose a batch of 32 samples to be labelled next. The size of the unlabelled samples in the plot is set to be proportional to their prediction entropy.

Uncertainty sampling chooses only samples very close to the current decision border, which is clearly visible in this example task, see Fig. 5.1: In the batch-mode setting only points on the linear decision border are chosen, which is the line of unlabelled points with the biggest size. The chosen points are very close to each other and cover only a very small part of the whole sample space, thus both diversity sampling and representative sampling are missing. Sequential uncertainty sampling chooses points with a little more diversity and representativeness.

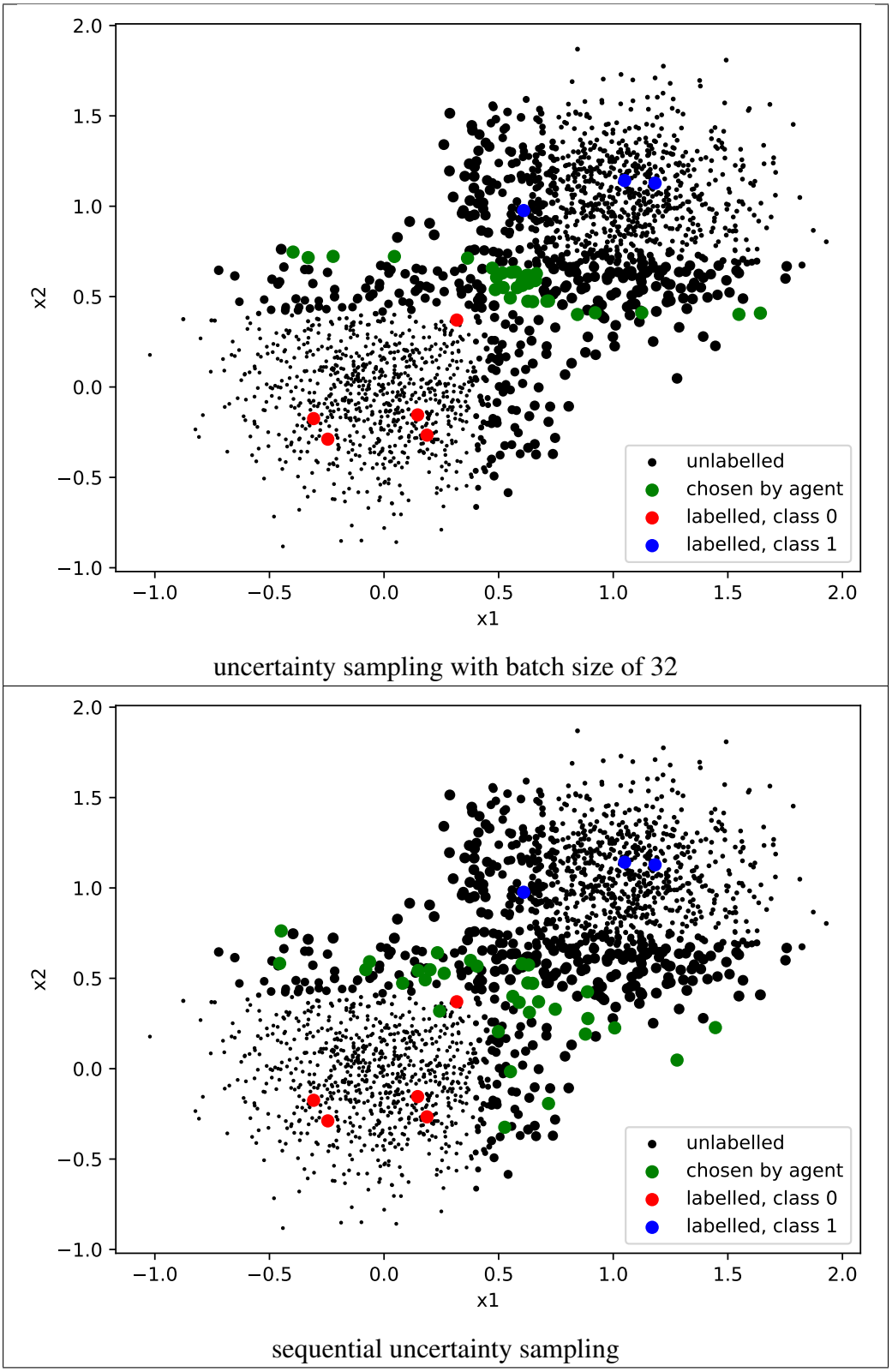

Figure 5.1.: Sample choices of uncertainty sampling on task with two Gaussian clouds: Uncertainty sampling fails to choose representative and diverse samples.

For the approach using Monte Carlo policy search, the beta vector was chosen such that the

β-parameters for the prediction uncertainty, minimum distance to the labelled and batch set and the 1st decil of the distance to the unlabelled set are 5, 5 and -5 respectively, all other β-parameters are set to zero. The corresponding choice of samples is shown in Fig. 5.2: The Monte Carlo agent clearly prefers to choose samples in the high-entropy regions, but mostly keeps a distance between the chosen samples and already labelled ones. A few samples are also chosen in regions with a low entropy to cover new parts of the unlabelled set. There are a few outliers chosen, e.g. the point at (-0.5, -0.6) is quite at the border of the Gaussian cloud.

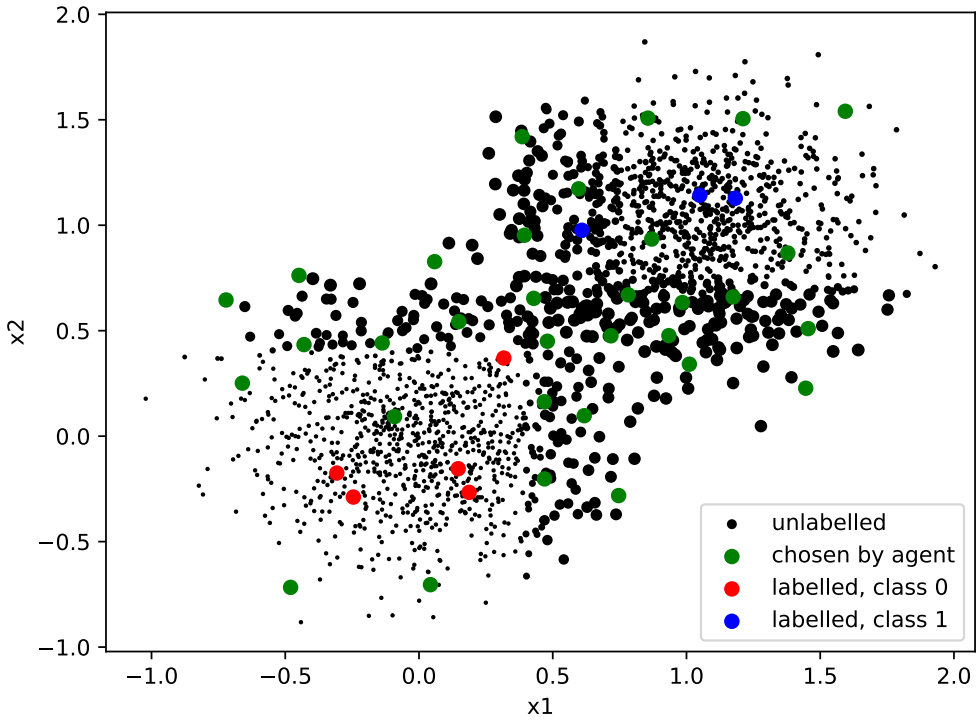

Figure 5.2.: Sample choices of batch-mode Monte Carlo agent on task with two Gaussian clouds: The agent chooses representative and diverse samples, while preferring uncertain regions.

For the uncertainty-weighed clustering approach, the base entropy weight factor was set to 2 or 5, the other entropy weight factors to zero. The corresponding choices of samples are shown in Fig. 5.3: The chosen samples cover the space of unlabelled samples well, while having a higher density in regions with a higher uncertainty. The comparison of both figures shows that the preference of regions with a higher uncertainty scales well with the entropy weight factor.

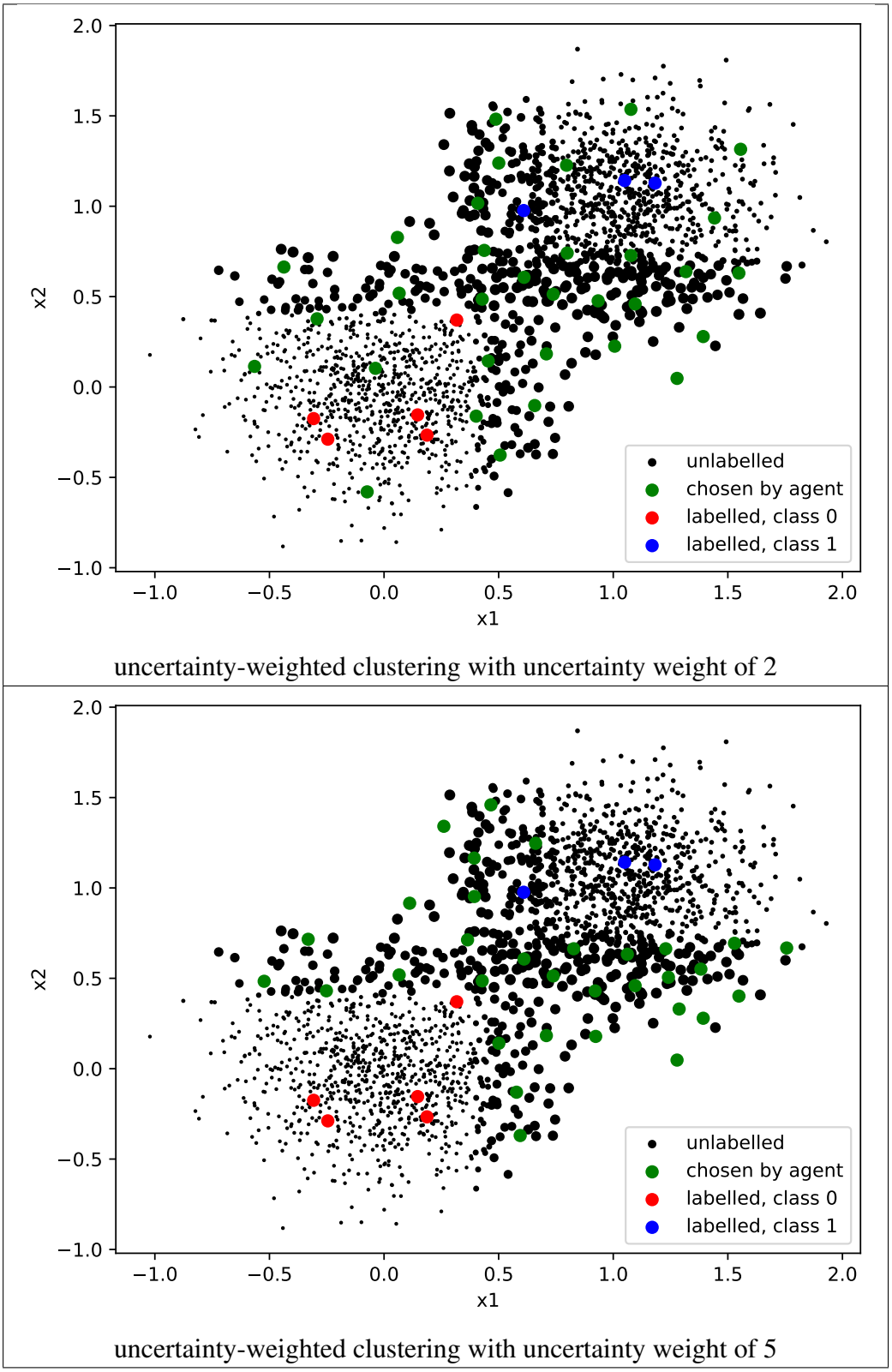

Figure 5.3.: Sample choices of uncertainty-weighted clustering agent on task with two Gaussian clouds: The agent distributes the samples to label across all samples, while preferring uncertain regions. The preference of uncertain regions is positively correlated with the uncertainty weight factor.

5.3. Parameters of agent using Monte Carlo policy search

The β-parameters for the agent using Monte Carlo policy search were optimized such that they maximize the objective function in Algorithm 7. The parameters are weights applied on the six different features used, namely the prediction entropy, the minimum and 10th percentile of the distances to the labelled and batch set and the 5th, 10th and 20th percentiles of the distances to the unlabelled set.

It was found that there is a very high correlation within the distance percentiles to the labelled and batch set and within the distance percentiles to the unlabelled set respectively. This high correlation of features caused instability problems of the optimizer, e.g. it happened that the β-parameter for one percentile was negative, while the others were positive. Thus, it makes more sense to interpret the sum of the corresponding beta parameters, i.e. the sum of β_2 and β_3 and the sum of β_4 to β_6 respectively. The best β-parameters learned on the 5 different training tasks are given in Table 5.2. As all features are normalized to have zero mean and unit variance, the absolute size of a parameter is proportional to the importance of the corresponding feature.

The sign of the beta parameters was found to align with the theoretical ideas from the enumeration 2.3: The β_1-parameter for the prediction entropy was always positive, thus the agent learned to prefer uncertain samples. The β_2+β_3-parameter sum for the distance percentiles to the labelled and batch samples was always positive, thus the agent performed diversity sampling. The β_4+β_5+β_6-parameter sum for the distances to the unlabelled set was always negative, thus representative samples were preferred.

feature	entropy	distance to labelled + batch			distance to unlabelled samples			
percentile		0th (min)	10th		5th	10th	20th	
parameter	β_1	β_2	β_3	β_2+β_3	β_4	β_5	β_6	β_4+β_5+β_6
training task(s)								
UCI	639.0	69.7	-21.2	48.5	57.8	-125.9	-4.2	-72.2
checker-board	20.6	17.7	4.9	22.6	-0.5	1.8	-4.8	-3.6
MNIST	8.6	3.8	4.8	8.6	-23.4	-37.1	16.0	-44.5
CIFAR10	2.6	4.0	14.4	18.3	-19.3	-16.4	10.2	-25.5
bAbI	50.7	22.6	-21.3	1.3	-1.9	-125.8	-17.7	-145.3

Table 5.2.: β-parameters learned on different training tasks by Monte Carlo agent: The best active learning strategy is highly dependent on the training task. Colours range from green (highest) to red (lowest).

The UCI dataset was the only dataset were uncertainty sampling performed better than random sampling and it is the dataset on which the highest β_1-parameter for the prediction

uncertainty was learned. Thus, the Monte Carlo agent has learned to rely strongly on the uncertainty sampling heuristic in a case where this heuristic performs well. The β-parameters 4 to 6 learned on the checkerboard task are close to zero and much smaller than the parameters for the entropy and diversity sampling, thus representative sampling is mostly unimportant for the checkerboard tasks. This can be well explained by the fact that the checkerboard tasks, where the sample points are uniformly distributed inside a square, does not contain any outliers. The β-parameters 4 to 6 learned on the vision task and even more on the question answering task have a very high absolute value, especially compared to the other β-parameters learned on the respective tasks, indicating that representative sampling is very important for these tasks.

The high difference between the β-parameters learned on different tasks indicates that there is no single best active learning strategy, but rather that the best active learning strategy depends highly on the dataset.

5.4. Parameters of uncertainty-weighted clustering agent

The three entropy weight factors according to Equation 3.5, the entropyWeightFactor_base, entropyWeightFactor_relBudget and entropyWeightFactor_relBudget_sqrt are optimized by training the clustering agent to give the best performance on the training tasks. The learned parameters are shown in Table 5.3.

On all tasks a positive base entropy weight was learned, except for the CIFAR10 task, for which it is approximately 0. Thus, it was learned to prefer uncertain samples. Just like for the Monte Carlo agent, the uncertainty importance in form of the β_1-parameter respectively the base entropy weight factor is highest when trained on the UCI tasks and lowest when trained on CIFAR10.

The entropy weights for the relative annotation budget and its square root are either 0 or positive for all tasks, thus uncertainty sampling becomes more important towards the end of an active learning run. One explanation for this is that that the higher number of labelled samples, and thus the better accuracy of the supervised learning model, increase the usefulness of uncertainty sampling.

training task(s)	**entropy weights**		
	base	**annotation budget**	$\sqrt{\textit{annotation budget}}$
UCI	47.28	-0.37	0.02
checkerboard	1.94	1.42	2.67
MNIST	2.18	1.69	1.84
CIFAR10	-0.25	2.86	1.34
bAbI	2.45	1.09	1.73

Table 5.3.: Parameters of entropy weights learned on different training tasks by clustering agent: The uncertainty weight factor is positively correlated to the relative performance of uncertainty sampling on the training task. Colours range from green (highest) to red (lowest).

5.5. Future research directions

There are many directions in which future research based on this thesis could be conducted. Promising directions are the following:

- The whole process of learning active learning has a huge number of design choices and hyperparameters. This space could be searched much more extensively. As an example, different training tasks or training tasks combinations could be used. Furthermore, more annotation batch sizes could be tried. The hyperparameters of the supervised learning models could be chosen not only such that they maximize the performance with 200 random samples, but rather with the labelled set currently chosen by the agent. There are many more design choices as learning active learning is a combination of supervised learning and reinforcement learning, not all of them can be named.
- Both the feature set of the Monte Carlo and the clustering agent is very small and does not include any non-linear combinations of features. Instead, more and different features and their combinations could be used. Furthermore, the diversity and representativeness sampling of the learning agents is based on euclidean distances. Many other distance metrics could be used, especially for categorical and integer data or features with diverging scales.
- The representation of images using tSNE features is only one possible feature reduction technique and has the disadvantage, that the structured nature of images is not used during the dimension reduction. It might be better to use convolutional autoencoders for the feature reduction of images or directly use a distance measure comparing the raw images.
- The Q-learning agent was chosen to be trained greedily, even though there are reinforcement learning methods like temporal difference learning or eligibility traces, which allow non-greedy behaviour.
- The clustering agent is using k-means clustering, even though many other clustering algorithms exist. They could be used instead, as long as they can be adapted to support fixed centers and weighted points. As an example, it might be useful to use an algorithm which assigns one unlabelled sample to multiple cluster centres, as unlabelled samples can be predicted based on multiple close samples and not only the single closest sample.
- The agent using Monte Carlo policy search and the one using uncertainty-weighted clustering could be combined, e.g. by using the agent with Monte Carlo policy search as initialization for the clustering agent and and then performing clustering only by a few number of steps.
- The generalization performance of the learning agents on other datasets, supervised learning models or metrics could be investigated. The choice of training and corresponding evaluation tasks (e.g. training on MNIST, evaluating on fashion-MNIST) in this thesis was mainly based on an intuition about a similarity of datasets, but was not evaluated scientifically. Another point of interest could be how well an agent trained on one supervised learning model, e.g. a CNN for classification performs on a task with another model, e.g. a CNN for semantic segmentation. Furthermore, one could try

how well a learned agent generalizes over different metrics like the accuracy, F1-score or more.

- The most promising research direction in our eyes is to try the agents using Monte Carlo policy search and uncertainty-weighted clustering in the warm-start active learning setting. These two agents have a low number of parameters compared to the number of samples in the dataset, and the parameters are not applied on samples itself, but rather on metrics highly dependent on the samples in the labelled set and the model trained on them. Thus, we assume that these parameters do not fit on any specific samples at all, but rather on general properties of the dataset. Given a sufficiently large initial training set, it should be possible to train a learning agent on this set which performs very well choosing the next samples to be annotated for the same task.

6. Conclusion and Outlook

The goal of this thesis, to develop an agent that learns active learning in the batch-mode setting, was fulfilled: The conceptual framework for modelling batch-mode active learning as a Markov decision process, in which a batch of samples is filled sequentially, worked well. It provided a clear and formalized definition of the problem, while keeping the action space one-dimensional and thus simple. Two of the three agents trained to find good policies given the MDP performed well: The agent using Monte Carlo policy search performed best across all tasks and performed at least on par with the best heuristic, but significantly better in most tasks. The clustering agent always performed worse than the Monte Carlo agent, but still better than random sampling across all tasks.

The relative performance of the agents and heuristics to each other varied greatly between tasks, as did the parameters learned by the learning agents when trained on different training tasks. Both indicate that there is no single best agent for every active learning tasks, but rather that the agent should be adapted to characteristics of the dataset and/or supervised learning model used. Such an adaptation to unknown tasks cannot be achieved by heuristics, which reinforces the need to replace them by agents learning active learning.

A high degree of adaptability in form of a policy with many parameters, such as the Q-learning agent, entails the need of many training samples and causes a high susceptibility to noise. It makes this approach computationally very expensive and hard to understand and causes a low reproducibility. On the other hand, a very low adaptability is also has a negative impact on the performance: The uncertainty-weighted clustering agent combines learning and an engineered policy and cannot learn to trade-off between diversity and representative sampling. It performs worse than the Monte Carlo approach.

The Monte Carlo agent has many properties explaining why it works so well: Compared to the Q-Learning agent it has a lower number of parameters, which allows to find the best parameters using black-box optimization. Thus, it has both lower computational costs and a much higher stability and reproducibility. Nevertheless, it allows to learn a trade-off between all features and has a high adaptability to tasks, making it superior to the clustering approach. Apart from its high performance, the Monte Carlo agent also has other advantages: Both its policy and training procedure are easy to understand and easy to implement, making the effort of applying it only slightly higher than when using a heuristic. Furthermore, it can easily be adapted to use more and other features, allowing to profit from the ongoing research on heuristics: It is sufficient to add any computed metric, on whose basis the heuristics decides, as an additional feature.
The main disadvantage of the Monte Carlo agent in comparison to heuristics, the high training costs, could be reduced in real scenarios by using the parameters learned on similar tasks as priors for the black-box optimization.

Even though we had to try many approaches, design options and parameter configurations, most of which were a dead end, this thesis achieved its goal and found an agent that can learn

active learning in the batch-mode setup. Apart from proposing such an agent it also gave an insight into the main ideas of active learning and why a combination of them is necessary for any well-working agent. Most importantly, it demonstrated the performance gains of learning active learning in the batch-mode setup using Monte Carlo policy search, which can be achieved consistently across many different datasets and supervised learning models.

A. Additionally

A.1. German abstract

Überwachte Lernmodelle arbeiten am besten, wenn sie auf viele Daten trainiert werden, allerdings ist das Labelling von Trainingsdaten in einigen Bereichen sehr kostspielig. Aktives Lernen zielt darauf ab, nur die informationsreichste Teilmenge von ungelabelten Daten für die Annotation auszuwählen und so Annotationskosten zu sparen. Es sind zwar mehrere Heuristiken für die Auswahl dieser Untergruppe entwickelt worden, doch ihnen fehlt die Anpassungsfähigkeit an verschiedene Aufgaben und Datensätze. Daher wurden sie vor kurzem durch Methoden ersetzt, die die beste Auswahl aus den Daten lernen. Dies wird durch die Formulierung des aktiven Lernens als Markov-Entscheidungsprozess und die Anwendung von Methoden des Reinforcement Learning erreicht. Die neuere Literatur zum aktiven Lernen deckt jedoch nur den sequentiellen Fall ab, während das aktive Lernen im Batch-Modus viele praktische Vorteile hat.
Mit der vorliegenden Arbeit soll diese Lücke geschlossen werden, indem ein Agent entwickelt und evaluiert wird, der aktives Lernen im Batch-Modus ermöglicht. Zunächst wird vorgeschlagen, aktives Lernen im Batch-Modus als eine Aufgabe zu modellieren, bei der der Batch sequentiell gefüllt wird. Dadurch wird der Entscheidungsraum für den Agenten exponentiell kleiner und somit das Training des Agenten erleichtert. Zweitens zeigt die Arbeit, dass ein Agent, der aktives Lernen im Batch-Modus lernt, in der Lage ist, Heuristiken über verschiedene Datensätze und überwachte Lernmodelle hinweg durchgängig zu übertreffen, was seine Anpassungsfähigkeit beweist.

A.2. Architecture of implementation

The implementation of applying an active learning agent on an active learning problem is divided into four modules. The goal of this architecture is to make it very easy both to implement new agents and to implement new tasks in which a supervised learning model is trained on a dataset. These modules are:

- Supervised learning task:
  The supervised learning task consists of a dataset and a machine learning model to train on a subset of this dataset.
- Active learning environment:
  The environment defines the Markov decision process including the step-function, observation space, action space and rewards for a given task.

- Active learning agent:
  An active learning agent can be applied on the Markov decision process defined by the environment, thus it has a policy choosing an action given an observation.
- Application handler:
  This class initializes the task, environment and agent, applies the agent iteratively on the active learning environment and saves the results.

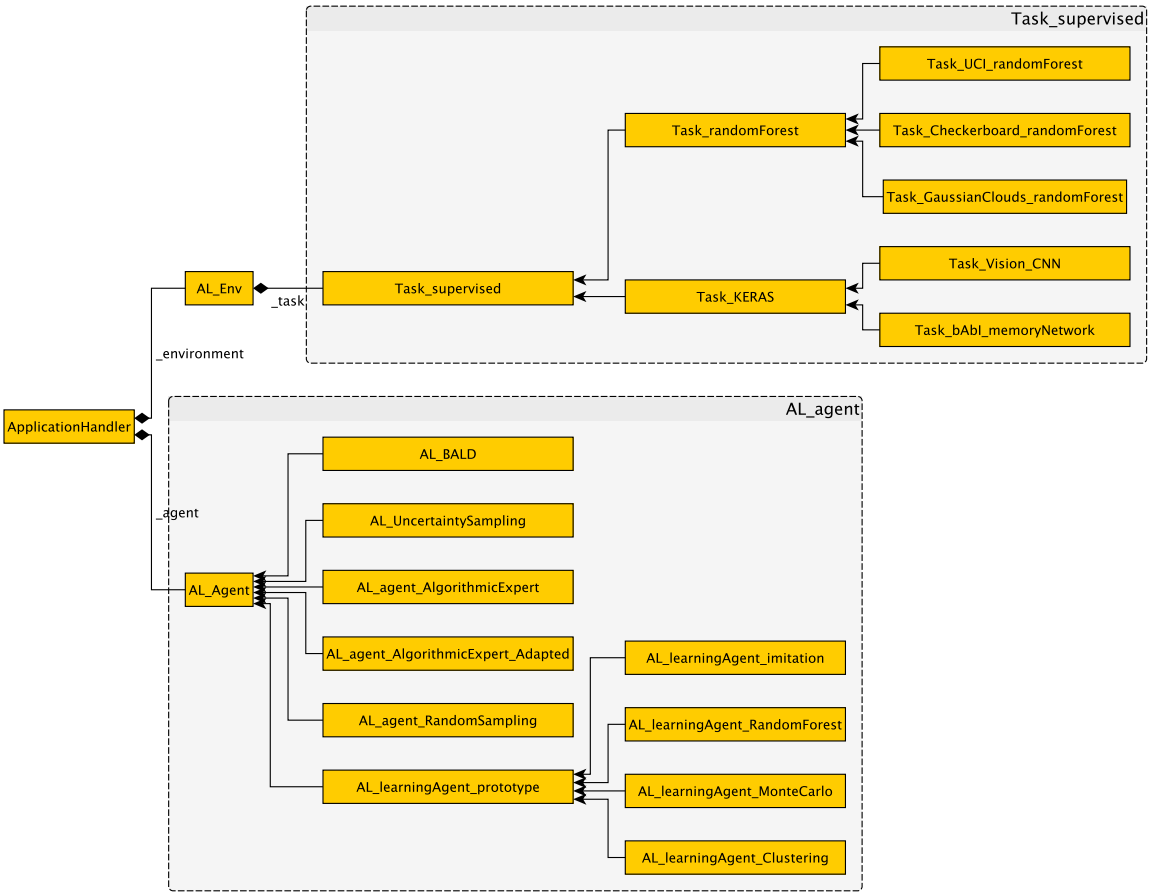

Figure A.1.: Architecture of active learning framework: The application handler contains both an agent and an environment defining the MDP, the environment itself contains a supervised learning task.

A.2.1. Implementation of supervised learning task

Each supervised learning task must be implemented as a class inheriting from the *Task_supervised* parent class. This class defines all methods the task must implement. It already implements some of them if they are independent of the specific task, others must be implemented in the child class.
Its most important method is the *trainOnBatch* method. It takes a set of labelled samples, defined by their IDs, as input and returns the loss and the accuracy on the validation set if the supervised learning model is trained on these samples. This method is needed to evaluate the choice of samples by the active learning agents.

Another very important method is the *getSampleInfo* method. It computes all the data about the labelled and unlabelled samples which is used later to define the observation space on whose basis the active learning agents decide which samples to label next. The other methods are mainly called by these two main methods, e.g. to compute parts of the features of the observation.

For implementing the algorithmic expert further methods are needed: One to estimate the improvement of the losses if the supervised learning model is trained on an additional sample and methods to get the current model representation and set it, which are needed to prevent that the actual model, and thus the observations, are updated.

The *Task_randomForest* class inherits from the *Task_supervised* parent class. It implements all methods needed according to the parent class which are dependent on the classifier used, but independent of the specific dataset used by a task. This includes implementing the random forest used as classifier and methods to get the predictions and prediction uncertainty of the unlabelled sample, which are used by the *getSampleInfo* method. Furthermore, it implements the *trainOnBatch* method used to train the classifier. Using this class, all tasks using a random forest as classifier only have to implement a single method for getting the dataset, as shown in the corresponding UML-diagram in Fig. A.2. This does not only make it easy to implement new tasks, but it also allow to change parameters of the classifier or adapt methods to get parts of the observation without needing to change them in the code for all tasks. Thus, it makes the development both faster and less error-prone.

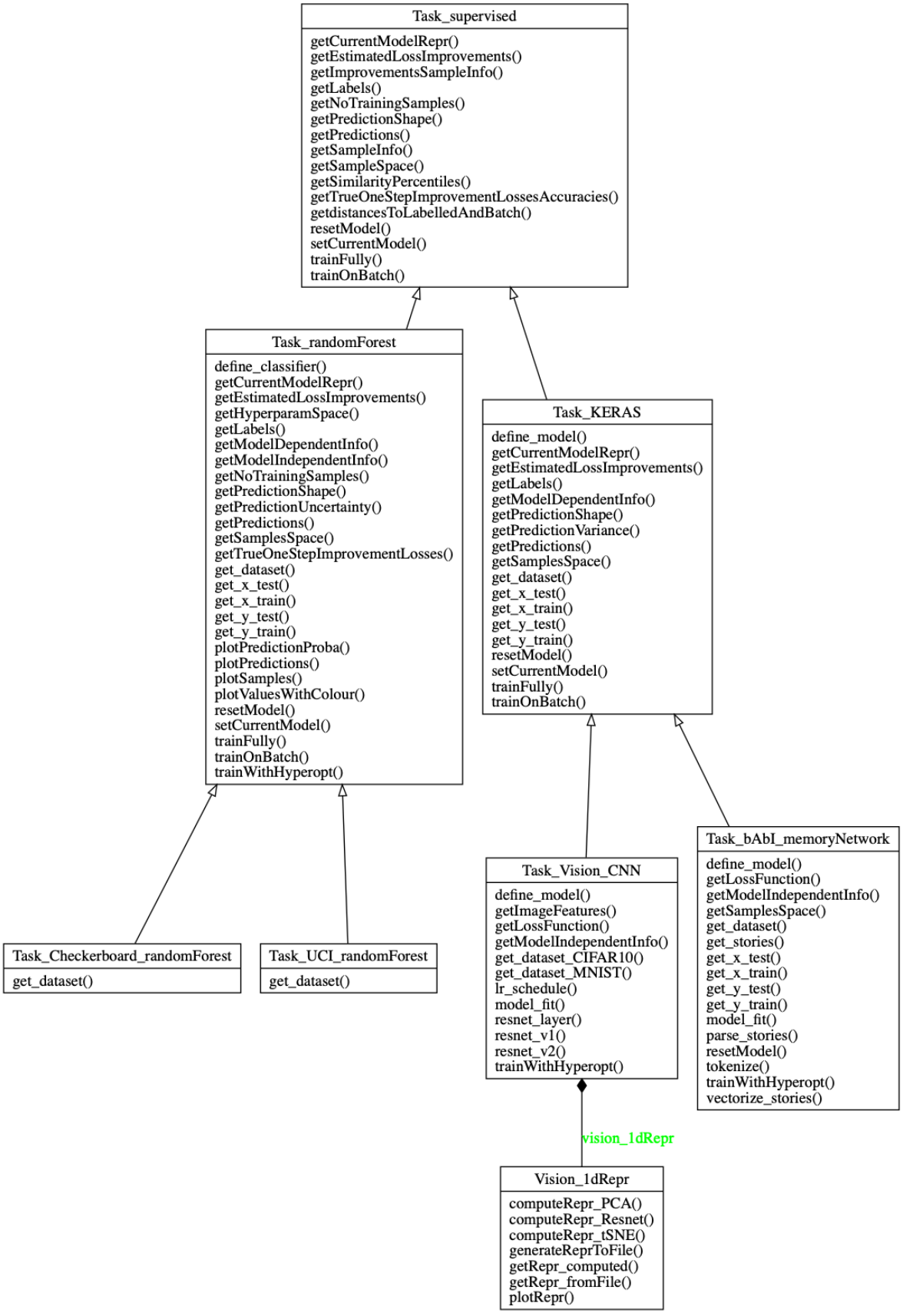

Figure A.2.: Architecture of supervised learning tasks: The inheritance allows to re-use code and quickly implement new tasks or datasets.

All checkerboard tasks are implemented as one class inheriting from the *task_supervised_randomForest* class and only need to implement the *get_dataset* to become a fully functional task. This also applies to the UCI tasks. The code to generate the datasets was taken from the github repositories of two papers by Konyushkova et al. [6], [59]. As both tasks share exactly the same supervised learning model, they only need to implement a single methods each, all other functionality is already implemented by the parent class.

Similar to the *task_supervised_randomForest* class, the *task_supervised_keras* class inherits from the *task_supervised* parent class and implements function which are the same for all classifiers implemented as keras models. As an example, the method to calculate the uncertainty of a keras classifier as the variance of its prediction using dropout is implemented in this class. The two tasks using this class only have to implement the methods to define the keras classifier and to get the dataset, as shown in the corresponding UML-diagram in Fig. A.3, which has the same advantages concerning fast and less error-prone software development. The calculation of the single-dimensional representations of the image data used in the *task_Vision_CNN* is done in the *Vision_1dRepr* class. Apart from the calculations itself, it also includes methods for writing the representation to files and reading them to prevent doing the costly calculations every time a new task is generated.

A.2.2. Implementation of active learning agents

All agents inherit from a parent class *AL_Agent*, which defines the arguments and return value of the *policy* method, which must be implemented by all agents. The non-learning agents, in particular the algorithmic expert, uncertainty sampling, random sampling and epsilon-greedy sampling only implement this method, as their policy is already pre-defined. The learning agents inherit additionally from the class *AL_learningAgent_prototype*, which defines methods for saving and loading an agent's internal learned model from a file or transforming the observation to be in a tensor instead of dictionary format.

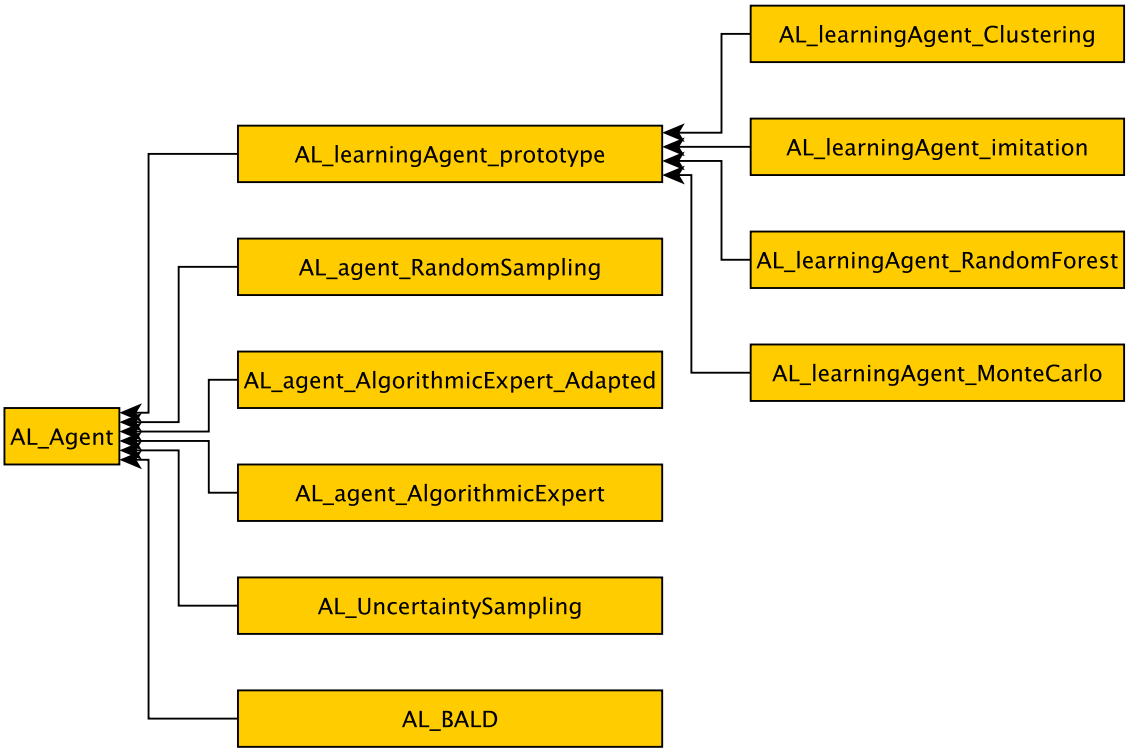

Figure A.3.: Architecture of active learning agents: The class *AL_Agent* defines the basic functionality needed, the class *AL_learningAgent_prototype* defines the additional functions needed for learning agents, thus all four learning agents inherit from it.

A.2.3. Implementation of training a learning agent

The training of the agents using Monte Carlo policy search or uncertainty-weighted clustering is quite straightforward: Define the objective function and apply the black-box algorithm on it. The training of a Q-learning agent and in particular the fitting of its Q-function to the generated samples in form of a regressions task, is implemented such that it takes four steps:

1. Loading of observations:
   First the observations are loaded from the corresponding *applicationHandlers* saved in files. It is possible to use training data from different *applicationHandlers* and/or to use them as validation data. The observations are still in the form of one dictionary per step of the MDP and do not include all possible features.

2. Preparation of training data:
   Next the training data is prepared: First the rewards are extracted out of the observations and normalized to have zero mean and unit variance. The features itself are calculated batch-wise on the fly by a data generator. In the observations, the features are in form of a dictionary and not all features are calculated yet, while the learning agents require one numpy array per sample. Thus, four calculations are needed: First, the three similarity measures of each sample's prediction and 1d-representation to the

mean one of the labelled and unlabelled set are calculated. Next the features per observation like the size of the labelled set are broadcast over the number of samples in the observation. Then all these features are concatenated to a 2d numpy-array with the samples on one axis and the features on the other. Last the mean values per feature are calculated and added as additional features to each sample. These steps all increase the memory usage, thus they are performed only when training the agent and not when saving the observations.
Another advantage of the usage of the data generator is that it allows to define and calculate additional features without needing to generate the underlying observations again. It further allows to shuffle the samples and to separate a subset of the data to set up another data generator for the validation set.

3. Training of agent:
   The agents all have a method allowing to train them given data generators for the features and the targets. While the neural networks using keras allow to use the data generators on-the-fly, the OLS regressor and random forest do not support this. Instead, they extract the features and targets from the data generator and train on all of them at once.

4. Saving of agent:
   In the last step, the Q-function estimated by the agents is saved in an appropriate form: The keras models are saved in the h5-format, the random forest regressor using jsonpickle and the parameters of the OLS-regressor as numpy array.

A.2.4. Implementation of procedures

There are three main procedures for the approach using Q-learning, which are all implemented as scripts: One for the generation of training data, one for training an agent on the training data saved in a file and one for applying agents on an active learning task. The generation of training data also needs to train an agent for its on-policy sampling and to perform an active learning task, but including saving the observations, thus these functions are implemented such that they can be used by all procedures. Both the application of a learning agent and the training of it need the function to calculate the features of the learning agent out of the observation, thus this function is also shared across both procedures. The relationship between the procedures is shown in Fig. A.4.

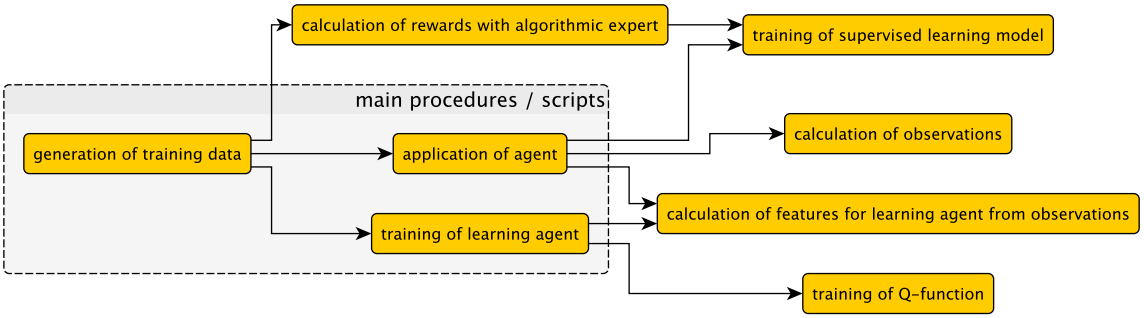

Figure A.4.: Hierarchy of main procedures for Q-learning and their compute-intensive subfunctions: The generation of training data is very costly, as it includes both the application of an agent on an active learning task and the training of a learning agent in the form of fitting its Q-function to generated training samples.

The approaches using Monte Carlo policy search or uncertainty-weighted clustering are easier, as they do not need any training samples and their agent can be defined by a few parameters: Both during training and application they perform active learning with a fixed policy defined by the fixed parameters. During training, several parameter combinations are tried sequentially to find the best one. This parameter combination was copied and hardcoded as default parameter combination for the evaluation tasks of the task it was trained on.

A.2.5. Reduction of computation time

Because of the high variance of applying an agent on a task and training an agent, it is necessary to repeat all steps of training and evaluating an active learning agent many times. Thus, it became necessary to optimize the code to reduce the computation time. There are four sources for the high computational cost, which are the subfunctions shown in Fig. A.4.

- Training of supervised learning model:
  Both the active learning process itself and the calculation of rewards with an algorithmic expert require to train a supervised learning model on a given dataset. Training this supervised learning model is quite compute-intensive, especially for the neural networks needing multiple epochs. One such training is necessary every time a batch is full (when applying an agent) or every time the algorithmic expert calculates a reward. Generating 50,000 training samples with the algorithmic expert thus needs 50,000 trainings of a supervised learning model. These steps are made faster by always performing multiple active learning runs in parallel and concatenating the generated training samples. As we used a 52-core, 104-thread server, we ran up to 104 active learning runs in parallel. To decrease the memory footprint, the random forest for the Q-function, which might be multiple GB big, is a shared variable for all active runs.
- The calculation of observations is also compute-intensive, e.g. predicting the label for all samples in the unlabelled set and calculating the uncertainty of this prediction. As the prediction is performed using a method of the supervised learning model and its corresponding library, this step is already optimized internally.

- The calculation of features for the learning agent from the observation includes the computation of distance percentiles of a sample and its prediction to those of the samples in the labelled and unlabelled set, making it also compute-intensive. In this step the computation time was reduced significantly by doing these calculations not in a loop, but always using vectorized functions applied on tensors. The vectorized computations are executed efficiently by numpy as it uses C-code instead of Python-code internally. Furthermore, only features which change in one step of the MDP are, computed newly. The other features are copied from the old observation and the row for the sample just chosen to be labelled is deleted.
- The training of the Q-function of a learning agents trains a random forest with about 1000 estimators on about 50000 training samples with about 70 features. This step is already internally parallelized as part of the scikit library, thus we did not optimize it further.

List of Figures

List of Tables

Bibliography

[1] B. Settles, "Active learning literature survey," University of Wisconsin–Madison, Computer Sciences Technical Report 1648, 2009.

[2] D. D. Lewis and W. A. Gale, "A sequential algorithm for training text classifiers," in *SIGIR'94*. Springer, 1994, pp. 3–12.

[3] M. Fang, Y. Li and T. Cohn, "Learning how to active learn: A deep reinforcement learning approach," *arXiv preprint arXiv:1708.02383*, 2017.

[4] M. Woodward and C. Finn, "Active one-shot learning," *arXiv preprint arXiv:1702.06559*, 2017.

[5] P. Bachman, A. Sordoni and A. Trischler, "Learning algorithms for active learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 301–310.

[6] K. Konyushkova, R. Sznitman and P. Fua, "Learning active learning from data," in *Advances in Neural Information Processing Systems*, 2017, pp. 4225–4235.

[7] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[8] K. Brinker, "Incorporating diversity in active learning with support vector machines," in *Proceedings of the 20th international conference on machine learning (ICML-03)*, 2003, pp. 59–66.

[9] S. C. Hoi, R. Jin and M. R. Lyu, "Large-scale text categorization by batch mode active learning," in *Proceedings of the 15th international conference on World Wide Web*, 2006, pp. 633–642.

[10] S. C. Hoi, R. Jin, J. Zhu and M. R. Lyu, "Batch mode active learning and its application to medical image classification," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 417–424.

[11] G. Schohn and D. Cohn, "Less is more: Active learning with support vector machines," in *ICML*, vol. 2, no. 4. Citeseer, 2000, p. 6.

[12] R. S. Sutton, A. G. Barto *et al.*, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[13] T. K. Ho, "Random decision forests," in *Proceedings of 3rd international conference on document analysis and recognition*, vol. 1. IEEE, 1995, pp. 278–282.

[14] O. Sener and S. Savarese, "Active learning for convolutional neural networks: A core-set approach," *arXiv preprint arXiv:1708.00489*, 2017.

[15] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[17] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," *arXiv preprint arXiv:1712.04621*, 2017.

[18] E. J. Bjerrum, "Smiles enumeration as data augmentation for neural network modeling of molecules," *arXiv preprint arXiv:1703.07076*, 2017.

[19] J.-Y. Zhu, T. Park, P. Isola and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[20] X. J. Zhu, "Semi-supervised learning literature survey," University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2005.

[21] K. Nigam, A. K. McCallum, S. Thrun and T. Mitchell, "Text classification from labeled and unlabeled documents using em," *Machine learning*, vol. 39, no. 2-3, pp. 103–134, 2000.

[22] D. Yarowsky, "Unsupervised word sense disambiguation rivaling supervised methods," in *33rd annual meeting of the association for computational linguistics*, 1995, pp. 189–196.

[23] A. Blum and T. Mitchell, "Combining labeled and unlabeled data with co-training," in *Proceedings of the eleventh annual conference on Computational learning theory*, 1998, pp. 92–100.

[24] D. Angluin, "Queries revisited," in *International Conference on Algorithmic Learning Theory*. Springer, 2001, pp. 12–31.

[25] J.-J. Zhu and J. Bento, "Generative adversarial active learning," *arXiv preprint arXiv:1702.07956*, 2017.

[26] S. Tong, *Active learning: theory and applications*. Stanford University USA, 2001, vol. 1.

[27] B. Settles, M. Craven and L. Friedland, "Active learning with real annotation costs," in *Proceedings of the NIPS workshop on cost-sensitive learning*. Vancouver, CA, 2008, pp. 1–10.

[28] S. Arora, E. Nyberg and C. P. Rosé, "Estimating annotation cost for active learning in a multi-annotator environment," in *Proceedings of the NAACL HLT 2009 Workshop on Active Learning for Natural Language Processing*. Association for Computational Linguistics, 2009, pp. 18–26.

[29] R. Greiner, A. J. Grove and D. Roth, "Learning cost-sensitive active classifiers," *Artificial Intelligence*, vol. 139, no. 2, pp. 137–174, 2002.

[30] A. Kapoor, E. Horvitz and S. Basu, "Selective supervision: Guiding supervised learning with decision-theoretic active learning." in *IJCAI*, vol. 7, 2007, pp. 877–882.

[31] W. Cai, M. Zhang and Y. Zhang, "Batch mode active learning for regression with expected model change," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 7, pp. 1668–1681, 2016.

[32] P. Munjal, N. Hayat, M. Hayat, J. Sourati and S. Khan, "Towards robust and reproducible active learning using neural networks," *arXiv*, pp. arXiv–2002, 2020.

[33] T. Scheffer, C. Decomain and S. Wrobel, "Active hidden markov models for information extraction," in *International Symposium on Intelligent Data Analysis*. Springer, 2001, pp. 309–318.

[34] C. E. Shannon, "A mathematical theory of communication," *Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[35] I. Dagan and S. P. Engelson, "Committee-based sampling for training probabilistic classifiers," in *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 150–157.

[36] I. Muslea, S. Minton and C. A. Knoblock, "Selective sampling with redundant views," in *AAAI/IAAI*, 2000, pp. 621–626.

[37] N. A. H. Mamitsuka *et al.*, "Query learning strategies using boosting and bagging," in *Machine learning: proceedings of the fifteenth international conference (ICML'98)*, vol. 1. Morgan Kaufmann Pub, 1998.

[38] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[39] M. McCallum and K. Nigam, "Employing em in pool-based active learning for text classification," in *Machine learning: proceedings of the fifteenth international conference (ICML'98)*, vol. 1. Morgan Kaufmann Pub, 1998, pp. 359–367.

[40] B. Settles, M. Craven and S. Ray, "Multiple-instance active learning," in *Advances in neural information processing systems*, 2008, pp. 1289–1296.

[41] N. Houlsby, F. Huszár, Z. Ghahramani and M. Lengyel, "Bayesian active learning for classification and preference learning," *arXiv preprint arXiv:1112.5745*, 2011.

[42] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, 2016, pp. 1050–1059.

[43] Y. Gal, R. Islam and Z. Ghahramani, "Deep bayesian active learning with image data," *arXiv preprint arXiv:1703.02910*, 2017.

[44] G. Wang, J.-N. Hwang, C. Rose and F. Wallace, "Uncertainty sampling based active learning with diversity constraint by sparse selection," in *2017 IEEE 19th International Workshop on Multimedia Signal Processing (MMSP)*. IEEE, 2017, pp. 1–6.

[45] L. J. Savage, "On rereading ra fisher," *The Annals of Statistics*, pp. 441–500, 1976.

[46] S. C. Hoi, R. Jin, J. Zhu and M. R. Lyu, "Semisupervised svm batch mode active learning with applications to image retrieval," *ACM Transactions on Information Systems (TOIS)*, vol. 27, no. 3, pp. 1–29, 2009.

[47] B. Demir, C. Persello and L. Bruzzone, "Batch-mode active-learning methods for the interactive classification of remote sensing images," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 49, no. 3, pp. 1014–1031, 2010.

[48] A. Kirsch, J. van Amersfoort and Y. Gal, "Batchbald: Efficient and diverse batch acquisition for deep bayesian active learning," in *Advances in Neural Information Processing Systems*, 2019, pp. 7024–7035.

[49] A. Liebgott, D. Boborzi, S. Gatidis, F. Schick, K. Nikolaou, B. Yang and T. Küstner, "Active learning for automated reference-free mr image quality assessment: analysis of

the influence of intra-batch redundancy on the number of required training samples," in *International Society for Magnetic Resonance in Medicine*, Jun. 2018.

[50] M. Liu, W. Buntine and G. Haffari, "Learning how to actively learn: A deep imitation learning approach," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 1874–1883. [Online]. Available: https://www.aclweb.org/anthology/P18-1174

[51] P. Dayan and Y. Niv, "Reinforcement learning: the good, the bad and the ugly," *Current opinion in neurobiology*, vol. 18, no. 2, pp. 185–196, 2008.

[52] M. Liu, W. Buntine and G. Haffari, "Learning to actively learn neural machine translation," in *Proceedings of the 22nd Conference on Computational Natural Language Learning*. Brussels, Belgium: Association for Computational Linguistics, Oct. 2018, pp. 334–344. [Online]. Available: https://www.aclweb.org/anthology/K18-1033

[53] S. Ravi and H. Larochelle, "Meta-learning for batch mode active learning," 2018.

[54] M. Minsky, "Steps toward artificial intelligence," *Proceedings of the IRE*, vol. 49, no. 1, pp. 8–30, 1961.

[55] A. Y. Ng and M. I. Jordan, "Shaping and policy search in reinforcement learning," Ph.D. dissertation, University of California, Berkeley Berkeley, 2003.

[56] K. Pearson, "Liii. on lines and planes of closest fit to systems of points in space," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 2, no. 11, pp. 559–572, 1901.

[57] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[58] T. Mikolov, K. Chen, G. S. Corrado and J. A. Dean, "Computing numeric representations of words in a high-dimensional space," May 19 2015, uS Patent 9,037,464.

[59] K. Konyushkova, R. Sznitman and P. Fua, "Discovering general-purpose active learning strategies," *arXiv preprint arXiv:1810.04114*, 2018.

[60] J. Bergstra, D. Yamins and D. D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," 2013.

[61] R. Z. Farahani and M. Hekmatfar, *Facility location: concepts, models, algorithms and case studies*. Springer, 2009.

[62] S. Lloyd, "Least squares quantization in pcm," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[63] A. P. Dempster, N. M. Laird and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[64] D. Arthur and S. Vassilvitskii, "k-means++: The advantages of careful seeding," Stanford, Tech. Rep., 2006.

[65] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[66] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[67] Y. LeCun, C. Cortes and C. Burges, "Mnist handwritten digit database," *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, 2010.

[68] H. Xiao, K. Rasul and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[69] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[70] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[71] J. Weston, A. Bordes, S. Chopra, A. M. Rush, B. van Merriënboer, A. Joulin and T. Mikolov, "Towards ai-complete question answering: A set of prerequisite toy tasks," *arXiv preprint arXiv:1502.05698*, 2015.

[72] S. Sukhbaatar, J. Weston, R. Fergus *et al.*, "End-to-end memory networks," in *Advances in neural information processing systems*, 2015, pp. 2440–2448.

[73] A. Irpan, "Deep reinforcement learning doesn't work yet," https://www.alexirpan.com/2018/02/14/rl-hard.html, 2018, accessed: 2020-09-07.

Declaration

Herewith, I declare that I have developed and written the enclosed thesis entirely by myself and that I have not used sources or means except those declared.

This thesis has not been submitted to any other authority to achieve an academic grading and has not been published elsewhere.

Stuttgart, 10.06.2020 Malte Ebner