Master thesis:
Increasing label efficiency in supervised classification for industrial application

Simon Bachhuber, Matrikelnummer: 1768044

Submitted in November 2020

UNIVERSITY REGENSBURG
Institute for Biophysics and physical Biochemistry

First assessor: Prof. Dr. Elmar Lang
Second assessor: Prof. Dr. Jaroslav Fabian

Contents

1. Introduction

In the last decade machine learning has evolved into a wide and popular field with many applications in real-world scenarios due to increasingly powerful hardware. With the recent development in Deep learning this trend is unlikely to stop since Deep learning is capable of modeling even highly complex dependencies/relationships. As an example, the introduction of convolutional neural networks has been a game changer in the field of computer vision which is of great importance to many other research areas.
However the majority of machine learning algorithms used in real-world application, including Deep learning, fall into the category of supervised learning (see appendix A for a two-page reminder) [20]. This means that in order to learn, the algorithm must be provided with sufficient number of input-output-pairs, and consequently the ability to obtain sufficient labeled data for modeling purposes has become one of the great challenges in a wide variety of learning problems. Labeled data is often expensive to receive, since it frequently involves human effort [1]. It is therefore desirable to minimize the cost associated with labeling data.
One option is to create labeled data cheaply using some non-human entity. This typically comes at the cost of an inferior label accuracy compared to data labeled by a human expert. Despite its lower label accuracy, after manipulating this pool of labeled data it may be capable of generalization and be used to solve the classification problem.
A second option is to rely on a human expert for labeling, but to at least choose the data to label in such a way that it increases the supervised algorithms performance as quickly as possible. Thus, the number of needed labels for a given accuracy goal is heavily reduced. This problem is tackled by the field of active learning.
In the following we will first develop both ideas theoretically and then apply active learning and label correction to toy examples to show a proof of concept. Next, the encoding of a real-world dataset is explained and then both ideas are deployed to increase the label efficiency in the learning associated to the real-world dataset.
Since the real-world dataset poses a classification problem, we restrict ourself to supervised classification and discard regression. Whereas regression deals with predicting continuous numbers (e.g. mileage of a car), classification predicts the membership to a finite number of categories (e.g. manufacturer of a car).
To summarize, the goal is to decrease the cost when labeling the required data for supervised classification and consequently make the arguably most prominent machine learning category more practical.

2. Active learning

A typical setup in machine learning is the classification of some input data to its corresponding class, e.g., assigning a handwritten number its numerical value where the input is an array of pixel values and the desired output is one of the ten classes. To achieve this a classifier must first be trained by providing input data with its respective class membership or label. But labeling data comes at a cost, as it often requires human labor.
While with passive learning unlabeled samples for the labeling process are chosen randomly, active learning refers to a certain strategy when picking the unlabeled samples. The goal is to train a classifier to the same capacity with fewer labels or, i.e., to learn faster when provided with the same amount of labeled training data resulting in a steeper learning curve. It is therefore most welcome when there is a large number of unlabeled data and labeling is expensive.
For example imagine an already working classifier that predicts class memberships with a certain accuracy. During its run time it will accumulate a pool of unlabeled data. To improve the performance the operator decides to invest resources to label some additional samples. It would be beneficial to choose those that will teach the classifier the most and result in the largest performance increase.
Active learning is shown to be effective in binary classification setups [9]. To motivate this idea of a strategy that leads to the reduction of necessary labels, consider this very simple example demonstrated in figure 2.1. Imagine dots on a line that have label "minus one" up to some point on the line and label "plus one" afterwards. This means that to achieve an error rate less than ϵ, $u = l = \mathcal{O}(1/\epsilon)$ unlabeled and labeled samples are necessary when the samples to label are chosen randomly (Let $X \sim \mathit{Binomial}(\epsilon, m)$, then $\overline{X} = \epsilon m \stackrel{!}{=} \mathcal{O}(1) \iff m = \mathcal{O}(1/\epsilon)$). Applying binary search for choosing the samples to label reduces the number of labels required to $l = \mathcal{O}(\log_2 1/\epsilon)$.
To see this, draw u randomly chosen *unlabeled* samples and align them on a line (this means sorting them by their "x-value"). Their hidden labels will be a sequence of minus ones and ones. The goal is to find the turning point between minus one and one. The sequence being sorted allows us to apply binary search. This means choosing the unlabeled sample in the middle, requesting its label and cutting the sequence in half. If the label is a minus one repeat this process on the "right" sequence, otherwise on the "left" until you find the turning point. This reduces the number of labels to find the turning point to $l = \log_2 u$. Since labeling all remaining unlabeled samples would not give additional information, we conclude that we achieve the same error rate as when all samples are labeled. Hence, we reach an error rate less than ϵ by requiring $u = \mathcal{O}(1/\epsilon)$

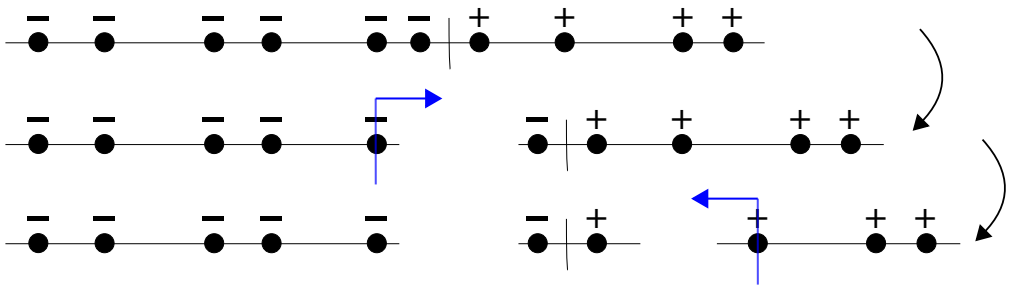


Figure 2.1.: Points on a line with label −1 up to some hidden value and +1 afterwards. The goal is find the hidden value or transition point up to some error. Choosing the samples to label following a binary search strategy leads an exponential speedup in the number of labels required to reach some error compared to choosing samples randomly. Note that the labels are only exposed for clarity.

unlabeled samples and $l = \mathcal{O}(\log_2 1/\epsilon)$ labeled samples. In the latter scenario carefully choosing the unlabeled samples for the label request reduces the amount of labels needed exponentially. So, to summarize, the goal of active learning is to exploit the currently available samples and labels in a way to effectively query samples that maximize the learning process of the model.

There are two major quantities that go into this decision making process, the informativeness and representativeness of a sample. The informativeness is a measure of the ability of a sample to reduce the uncertainty of a model, whereas representativeness measures how well an instance represents all other input patterns and is therefore necessary in order to preserve the underlying distribution $\mathcal{D}$ [15].
Many active learning strategies use either informativeness in the form of query-by-committee [28, 7, 13] or uncertainty sampling [17, 16, 30, 2] or representativeness through clustering methods [21, 8] or density measurements [34, 10]. Other active learning algorithms combine both criteria either in an ad hoc approach [10, 11], risk minimization [32] or min-max based active learning [15]. The term ad hoc refers to instantaneously calculating the informativeness and/or representativeness of each sample isolated.
Additionally, there are also performance-based strategies that try to directly estimate the accuracy increase of a classifier and choose samples accordingly [1].
Finally, there are adaptive learning algorithms designed to combine different strategies and automatically determine the weighting between them, their core idea typically revolves around solving a Multi-armed bandit problem [14, 23, 5].

The landscape of active learning mainly consists of two subsets, pool-based active learning and on-line active learning. In the following we talk about them separately.

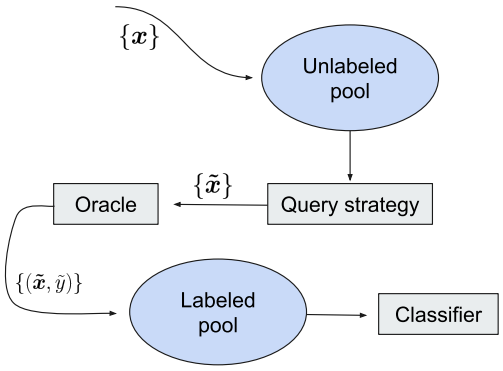


Figure 2.2: Schematic representation of the components involved in pool-based active learning. All unlabeled data is stored in the unlabeled pool. The query strategy then suggests which sample to label next and sends it to the oracle to receive its label. The labeled sample then gets added to labeled data and can be used to train a supervised classifier. The combination of query strategy and classifier is referred to as active learner.

2.1. Pool-based active learning

Pool-based active learning works, as the name suggests, by accumulating all unlabeled data inside one large pool/set and maintaining a ranking in which order the samples are to be labeled. To make the notion of data accumulated in sets more rigorous we continue by discussing some notation.

2.1.1. Preliminaries

Notation:
- Scalar: x, X
- Vector: $\boldsymbol{x}$ with entries x_i
- Array/Tensor: $\mathbf{X}$
- Function: $\mathcal{X}$, $\mathit{Function}$
- Set: $\mathcal{X}$

Suppose we are given N samples denoted as $\mathcal{S} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\}$ that are divided into $\mathcal{L} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), ..., (\boldsymbol{x}_L, y_L)\}$ a set of L labeled instances (the labeled pool) and $\mathcal{U} = \{\boldsymbol{x}_{L+1}, \boldsymbol{x}_{L+2}, ..., \boldsymbol{x}_N\}$ a set of $U = N - L$ unlabeled samples (the unlabeled pool), where $\boldsymbol{x}_n \in \mathbb{R}^D$ is a D-dimensional vector. We denote by $\mathcal{Z}$ the set of possible labels. The N samples, and labels $y_l \in \mathcal{Z}$ are independent and identically distributed (i.i.d.) according to an underlying distribution $\mathcal{D}$.

2.1.2. Components

All components that make up pool-based active learning are displayed in figure 2.2.
The general structure is as follows: All available, unlabeled data is stored in the unlabeled pool $\mathcal{U}$. Whenever an additional unlabeled sample should be labeled the query strategy decides which sample out of the unlabeled pool gets labeled. This process is called *querying*. The query strategy works by assigning every element of $\mathcal{U}$ an utility score and

outputs the best performing sample. As such the query strategy consists of a protocol to calculate an utility score and whether this measure is to min- or maximize, see algorithm 1 for an example query strategy that chooses the sample with the smallest feature/input vector. Often this protocol involves the classifier used to solve the classification problem and we refer to the active learner as the classifier combined with the query strategy.
After a sample to label is chosen it gets sent to the oracle. We denote the oracle as the instance that given the unlabeled sample provides us with the corresponding label. This task often involves a human expert. As a function the oracle is given by

$$\mathcal{O} : \mathbb{R}^D \to \mathcal{Z}$$

$$\boldsymbol{x}_n \to y_n = \mathcal{O}(\boldsymbol{x}_n) \text{ w.r.t. distribution } \mathcal{D}.$$

Afterwards the, now labeled, sample is added to the labeled pool $\mathcal{L}$ and the labeled pool can be used to train the classifier. We repeat this process until $\mathcal{U}$ is empty or some different criterion is fulfilled, e.g., the classifier reaches the required loss. This process is summarized in algorithm 2.

Before moving on we quickly mention the terms label and sample complexity.
Label and sample complexity describe the number of labeled samples (= number of labels) and the number of all samples (so with and without label) necessary to train the classifier to some point. For example, randomly choosing the samples to label gives an equal label and sample complexity as opposed to a query strategy as defined above where the sample complexity is N after the first query (since the query strategy needs access to all unlabeled data immediately). Anyhow, we hope to improve label complexity.

Algorithm 1 Example of a query strategy

Input: Unlabeled samples $\mathcal{U}$
Output: $\boldsymbol{x}_s \in \mathcal{U}$ # $\boldsymbol{x}_{\text{s(tar)}}$

1: $\mathcal{U}(\boldsymbol{x}_u) \coloneqq \|\boldsymbol{x}_u\|_2 \quad \forall \boldsymbol{x}_u \in \mathcal{U}$ # $\|.\|_2$ denotes Euclidean norm
2: $\boldsymbol{x}_s = \arg\min_{\boldsymbol{x}_u \in \mathcal{U}} \mathcal{U}(\boldsymbol{x}_u)$
3: **return** $\boldsymbol{x}_s$

2.1.3. The major flaw: Sampling bias

The approach as described above will have serious flaws, e.g., while randomly choosing the samples to label will preserve the distribution $\mathcal{D}$ from where the samples were drawn, introducing a query strategy will lead to samples that are drawn from a distribution induced by the query strategy. This might lead to a poor generalization ability of the model, since it no longer takes the distribution $\mathcal{D}$ into account [32]. We call this phenomenon *sampling bias*, i.e., the empirical probability density function (pdf) $\hat{\mathcal{D}}$ estimated using the queried samples strongly deviates from the unknown, true pdf $\mathcal{D}$.
This leads to the following, related problem - the labeled samples influence the choice of the next possible candidates for labeling. This means that when starting the active

Algorithm 2 Pool-based active learning

Input: Classifier $\mathcal{F}$ trained on $\mathcal{L}$, unlabeled samples $\mathcal{U}$, labeled samples $\mathcal{L}$
Output: $\mathcal{F}$ with $\mathcal{L}(\mathcal{F}) < \delta$

while $\mathcal{U} \neq \{\}$ **and** $\mathcal{L}(\mathcal{F}) > \delta$ **do**
  $\boldsymbol{x}_s = Q(\mathcal{F}, \mathcal{U}, \mathcal{L})$
  $y_s = O(\boldsymbol{x}_s)$
  $\mathcal{L}, \mathcal{U} = \mathcal{L} \cup \{(\boldsymbol{x}_s, y_s)\}, \mathcal{U} \setminus \boldsymbol{x}_s$
  retrain $\mathcal{F}$ on $\mathcal{L}$
  update $\mathcal{L}(\mathcal{F})$
end while
return $\mathcal{F}$

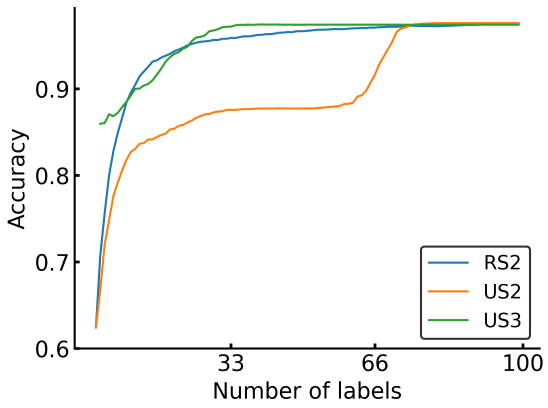


Figure 2.3.: Demonstrating the negative effect of starting an exploit heavy query strategy with limited labels initially. We compare either drawing samples randomly (RS) to querying the samples with highest entropy (US, section 2.1.4.2). The different plots RS2, US2 and US3 have as initial data two,two and three samples, respectively with every class being present at most once. The plots are obtained using a linear SVM on the Iris dataset and averaged over 500 runs.

learner with few labeled samples the initial sample bias will lead potentially to an even bigger bias [15]. This is demonstrated in figure 2.3 using an exploit-heavy query strategy such as uncertainty sampling, see section 2.1.4.2, with very few labeled samples initially. We say a query strategy is exploit-heavy if it is prone to induce sampling bias. The term exploit-heavy stems from the dilemma of balancing exploration (expanding your knowledge) versus exploitation (using your knowledge).
The key observation in this section is that *sampling introduces bias.*

2.1.4. Query strategies

As already mentioned the query strategy consists of an utility score and whether this score is to min- or maximize. Obviously there are arbitrarily many such choices and in the following we discuss some possible query strategies that, based on some superficial intuition, are assumed to improve label complexity. For completeness we note that we refer to all query strategies in this section as base query strategies for reasons that are not yet obvious.

2.1.4.1. Random sampling (RS)

This query strategy chooses a sample $\boldsymbol{x}_u \in \mathcal{U}$ to label at random. Sampling randomly corresponds to not having a strategy and is therefore our baseline. We also refer to sampling randomly as passive learning in contrast to active learning (actively following a strategy when choosing the samples to label). The goal is to reach a better performing classifier when choosing samples to label according to some policy as opposed to choosing randomly given some fixed number of labels in both cases.

2.1.4.2. Uncertainty sampling (US)

US is supposed to query informative samples. It defines an informative sample through the classifier's confidence in the sample's label prediction.
Intuition dictates that the classifier is the least certain for samples close to a decision boundary. Obviously those are also the samples with the most information in the sense that obtaining the labels of the samples that are closest to the decision boundary lead to the highest "resolution" of the true, unknown decision boundary. E.g., you may recall the introductory example of classifying dots on a line with an unknown label transition point.
So, we are interested in the points where the classifier is most uncertain of. There are different measures of assigning a discrete probability distribution an amount of uncertainty. Four possible choices are

- least confident: $\mathcal{S}(\boldsymbol{p}_{\boldsymbol{x}_i}) = 1 - \max_{z \in \mathcal{Z}} \; p_{\boldsymbol{x}_i, z}$,
- gini: $\mathcal{S}(\boldsymbol{p}_{\boldsymbol{x}_i}) = \sum_{z \in \mathcal{Z}} p_{\boldsymbol{x}_i, z}(1 - p_{\boldsymbol{x}_i, z})$,
- entropy: $\mathcal{S}(\boldsymbol{p}_{\boldsymbol{x}_i}) = \sum_{z \in \mathcal{Z}} -p_{\boldsymbol{x}_i, z} \; \ln p_{\boldsymbol{x}_i, z}$,
- margin: $\mathcal{S}(\boldsymbol{p}_{\boldsymbol{x}_i}) = \max_1 \; \boldsymbol{p}_{\boldsymbol{x}_i} - \max_2 \; \boldsymbol{p}_{\boldsymbol{x}_i}$ # $\max_n \; \boldsymbol{a}$ is the n-th largest element of $\boldsymbol{a}$

where $\boldsymbol{p}_{\boldsymbol{x}_i}$ is the vector of class membership probabilities (this vector represents a discrete probability distribution or probability mass function) belonging to sample $\boldsymbol{x}_i$ to be in one of the possible classes. Remember that $\mathcal{Z}$ is the set of all possible labels. If not noted differently, we use $\mathcal{S}$ as entropy.

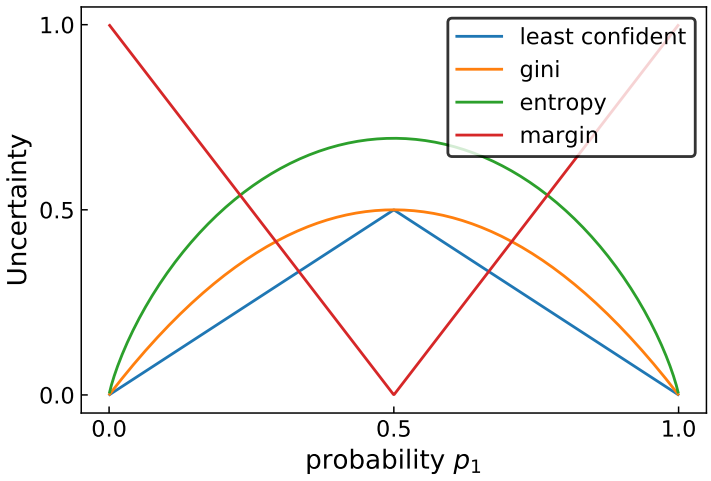


Figure 2.4.: Different uncertainty measures in binary classification, hence $p_1 + p_2 = 1$. We find $p_1 = p_2 = 0.5$ by maximizing all measures but "margin" and by minimizing "margin". In binary classification ordering samples after uncertainty will result in the same ranking for all uncertainty measures, in that sense all measures are equivalent.

Therefore, we need a classifier that is able to predict class membership probabilities. We denote this function by $\mathcal{P}$ and let

$$\boldsymbol{p}_{\boldsymbol{x}_i} := \mathcal{P}(\mathcal{F}, \boldsymbol{x}_i). \tag{2.1}$$

Every sample belongs to one of the possible classes and therefore it holds

$$\sum_{z \in \mathcal{Z}} p_{\boldsymbol{x}_i, z} = 1 \tag{2.2}$$

where $p_{\boldsymbol{x}_i,z}$ is the probability, predicted by classifier $\mathcal{F}$, that sample $\boldsymbol{x}_i$ has label $y_i = z$. All measures but "margin" share the property of being zero when the classifier is most certain and increase as uncertainty increases. Hence, we choose the sample that *minimizes* "margin" and *maximizes* else.
The strategy is summarized in algorithm 3.

2.1.4.3. Representative sampling (ReS)

ReS, as the name suggests, queries representative samples. A sample is representative if there are many other samples close (closeness through p-norm) in input space.
As already mentioned in section 2.1.3 simply sorting by heterogeneity, so by uncertainty, has its drawbacks in terms of loosing information about the distribution from where the samples were drawn initially.
One way of counteracting the influence of the query strategy on the distribution of drawn

Algorithm 3 Uncertainty sampling

Input: Classifier $\mathcal{F}$, unlabeled samples $\mathcal{U}$, labeled samples $\mathcal{L}$
Parameters: $\mathcal{S} \in \{\text{least confident}, \text{gini}, \text{margin}, \text{entropy}\}$
Output: $\boldsymbol{x}_s \in \mathcal{U}$

train $\mathcal{F}$ on $\mathcal{L}$
calculate class probabilities $\boldsymbol{p}_{\boldsymbol{x}_u} \forall \boldsymbol{x}_u \in \mathcal{U}$
if $\mathcal{S}$ is margin **then**
    $\boldsymbol{x}_s = \arg\min_{\boldsymbol{x}_u \in \mathcal{U}} \mathcal{S}(\boldsymbol{p}_{\boldsymbol{x}_u})$
else
    $\boldsymbol{x}_s = \arg\max_{\boldsymbol{x}_u \in \mathcal{U}} \mathcal{S}(\boldsymbol{p}_{\boldsymbol{x}_u})$
end if
return $\boldsymbol{x}_s$

samples, i.e., to get this distribution closer to the one that would emerge when choosing samples randomly, is by introducing density measures or clustering methods.
When solely relying on a representative measure as a utility score, the idea is that the classifier performs better if he first learns samples that are the most representative of all other samples. So, we need a similarity measure and we say that two samples $\boldsymbol{x}_i \in \mathcal{S}$ and $\boldsymbol{x}_j \in \mathcal{S}$ are similar if they are close in input space $\mathbb{R}^D$. So we choose a metric to quantify "close", e.g.,

- Manhattan: $\mathcal{M}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{d=1}^{D} |x_{i,d} - x_{j,d}|$,
- Euclidean: $\mathcal{M}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{\sum_{k=d}^{D} (x_{i,d} - x_{j,d})^2}$,
- Maximum: $\mathcal{M}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \max_{d=1..D} |x_{i,d} - x_{j,d}|$.

These metrics are all deduced from a p-norm with $p = 1, 2, \infty$ respectively. They are displayed in figure 2.5.
We define the density of a sample $\boldsymbol{x}_i$ in a given sample space $\mathcal{A}$ as

$$\rho_{\mathcal{A}}(\boldsymbol{x}_i, \epsilon) = \sum_{\substack{\boldsymbol{x}_a \in \mathcal{A} \\ \mathcal{M}(\boldsymbol{x}_i, \boldsymbol{x}_a) \leq \epsilon}} \frac{1}{A} \tag{2.3}$$

where $A = \sum_{\boldsymbol{x}_a \in \mathcal{A}} 1$ is the number of samples in $\mathcal{A}$ and ϵ is a hyperparameter. In words - what is the probability for a random sample $\boldsymbol{x}_a \in \mathcal{A}$ to not have a greater distance than ϵ to $\boldsymbol{x}_i$.

When computing ρ for more than one sample in an iterative labeling process it is beneficial to first compute the 2-point distance matrix $\mathbf{M}$ where

$$M_{i,j} \coloneqq \mathcal{M}(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{2.4}$$

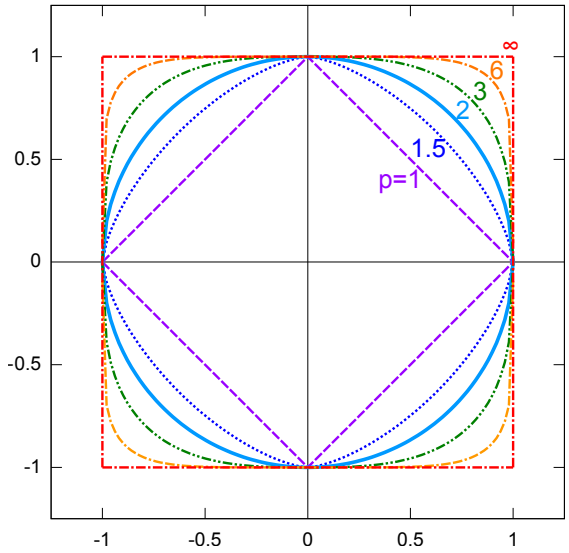


Figure 2.5.: Unit circle for different p-norm in two dimensions [25].

is a symmetric matrix with zero as diagonal elements.
We are mostly interested in $\rho_{\mathcal{S}}$ as it reflects the overall representativeness of a sample and $\rho_{\mathcal{L}}$ as the ability of a sample to represent the labeled pool. Consequently, two promising query strategies are to *maximize* $\rho_{\mathcal{S}}$ to find representative samples or to *minimize* $\rho_{\mathcal{L}}$ to reduce the redundancy in the training data.
Finally, representative sampling is condensed in algorithm 4.
Remark: $\rho_{\mathcal{S}} \to \mathcal{D}$ as $N \to \infty, \epsilon \to 0$ if samples are drawn i.i.d.

Algorithm 4 Representative sampling

Input: Unlabeled samples $\mathcal{U}$, labeled samples $\mathcal{L}$
Parameters: $\mathcal{R} \in \{\mathcal{L}, \mathcal{U}, \mathcal{S}\}$, $\epsilon > 0$, $g \in \{\text{high}, \text{low}\}$, metric $\mathcal{M}$
Output: $\boldsymbol{x}_s \in \mathcal{U}$

calculate and store $M_{n,n'} = \mathcal{M}(\boldsymbol{x}_n, \boldsymbol{x}_{n'}) \quad \forall n, n' = 1 \to N$
calculate $\rho_{\mathcal{R}}(\boldsymbol{x}_u) = \sum_{\substack{\boldsymbol{x}_r \in \mathcal{R} \\ M_{u,r} \leq \epsilon}} 1 \quad \forall \boldsymbol{x}_u \in \mathcal{U}$ # normalisation by $\frac{1}{R}$ is optional
if g is high **then**
    $\boldsymbol{x}_s = \arg\max_{\boldsymbol{x}_u \in \mathcal{U}} \rho_{\mathcal{R}}(\boldsymbol{x}_u)$
else
    $\boldsymbol{x}_s = \arg\min_{\boldsymbol{x}_u \in \mathcal{U}} \rho_{\mathcal{R}}(\boldsymbol{x}_u)$
end if
return $\boldsymbol{x}_s$

2.1.4.4. Mean distance sampling (MdS)

MdS is similar to ReS and also tries to find representative samples. A sample is representative if the average distance to another sample is small.
It is intuitive to assume that, if a sample is surrounded by an above average number of close-by samples, i.e., if it is in a high density region of data space then it will have a lower average distance w.r.t. all data points than a sample in a low density region of data space.
Note that to query samples in a *high* density region we look for samples with a *low* average distance.
MdS is summarized in algorithm 5.

Algorithm 5 Mean distance sampling

Input: Unlabeled samples $\mathcal{U}$, labeled samples $\mathcal{L}$
Parameters: $\mathcal{R} \in \{\mathcal{L}, \mathcal{U}, \mathcal{S}\}$, $g \in \{\text{high}, \text{low}\}$, metric $\mathcal{M}$
Output: $\boldsymbol{x}_s \in \mathcal{U}$

calculate and store $M_{n,n'} = \mathcal{M}(\boldsymbol{x}_n, \boldsymbol{x}_{n'}) \quad \forall n, n' = 1 \to N$
calculate $N = \sum_{\boldsymbol{x}_j \in \mathcal{R}} 1$
calculate mean distances $\boldsymbol{d}$ where $d_{\boldsymbol{x}_u} = \frac{1}{R} \sum_{\boldsymbol{x}_r \in \mathcal{R}} M_{u,r} \quad \forall \boldsymbol{x}_u \in \mathcal{U}$
if g is high **then**
    $\boldsymbol{x}_s = \arg\max_{\boldsymbol{x}_u \in \mathcal{U}} d_{\boldsymbol{x}_u}$
else
    $\boldsymbol{x}_s = \arg\min_{\boldsymbol{x}_u \in \mathcal{U}} d_{\boldsymbol{x}_u}$
end if
return $\boldsymbol{x}_s$

2.1.4.5. Nearest neighbour criterion (NNC)

NNC is designed to find representative samples and is taken from [35].
It finds representative samples by introducing a measure that quantifies how well the unlabeled pool is represented by the labeled pool - the lower the better. The query strategy then chooses the sample that *minimizes* this measure after the sample has been shifted from the unlabeled pool to the labeled pool.
We start by defining the total nearest neighbour distance as

$$\mathcal{N}(\mathcal{L}, \mathcal{U}) \coloneqq \sum_{\boldsymbol{x}_u \in \mathcal{U}} \min_{\boldsymbol{x}_l \in \mathcal{L}} \|\boldsymbol{x}_u - \boldsymbol{x}_l\|_2 \tag{2.5}$$

where $\|.\|_2$ denotes the Euclidean norm.
The function $\mathcal{N}$ is a direct measure of dissimilarity between the labeled and unlabeled pool. It obtains a small value when every unlabeled point has a labeled samples close-by and vanishes when the labeled pool contains every point of the unlabeled pool. This

justifies its interpretation as a measure of (dis)similarity. The lower the $\mathcal{N}$, the more representative is the labeled pool of the unlabeled pool.
Therefore, the query strategy chooses the sample such that $\mathcal{N}$ reaches a minimum after the sample has shifted from the unlabeled to the labeled pool, i.e., such that after having added one sample to the labeled pool it is now most representative of the remaining unlabeled pool. This means the criterion is given by

$$\boldsymbol{x}_s = \underset{\boldsymbol{x}_u \in \mathcal{U}}{\arg\min}\ \mathcal{N}(\mathcal{L} \cup \{\boldsymbol{x}_u\}, \mathcal{U} \setminus \{\boldsymbol{x}_u\}). \tag{2.6}$$

Note that, NNC rearranges the order in which the samples are to be labeled and this order does not depend on any labels. Hence, NNC delivers a static permutation of the unlabeled samples that can be pre-computed. This is in contrast to, e.g., US.

2.1.4.6. Density weighted uncertainty sampling (DwUS)

DwUS is meant to query informative and representative samples and is adopted from [27]. We find informative samples through US, see section 2.1.4.2. For representative samples we use a new approach that defines a representative sample as a sample similar to a cluster centroid.
In more details, we find the representative samples in the following way - first, given a dataset we assume that the cluster centroids that result from K-Means clustering are its most representative points (note that these points do not have to be part of the dataset). Taking into consideration that a set of cluster centroids are by definition the set of points that when every sample in the dataset is represented by the closest point out of the set of cluster centroids, then this set of cluster centroids minimizes the mean Euclidean distance between a point out of the dataset and its representation. This makes the assumption quite natural. Next, we say a sample is representative if it is similar to a cluster centroid. The similarity of two data points is defined as a normalized kernel function between the data points

$$\hat{\mathcal{K}}(\boldsymbol{x}_i, \boldsymbol{x}_j) \coloneqq \frac{\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\sqrt{\mathcal{K}(\boldsymbol{x}_i, \boldsymbol{x}_i)\,\mathcal{K}(\boldsymbol{x}_j, \boldsymbol{x}_j)}} \tag{2.7}$$

where $\mathcal{K}$ is an arbitrary kernel function.
Figure 2.6 shows the value of different kernel functions in a two-dimensional input space where the reference point is the cross in the center of the sub figures.
Then, the utility score that this query strategy *maximizes* is the product of an uncertainty measure and a similarity measure.
Algorithm 6 summarizes the query strategy DwUS.

2.1.4.7. Cluster margin sampling (CMS)

CMS is designed to query informative samples. It defines an informative sample as a sample close to a decision boundary but, in contrast to US, it does not require any

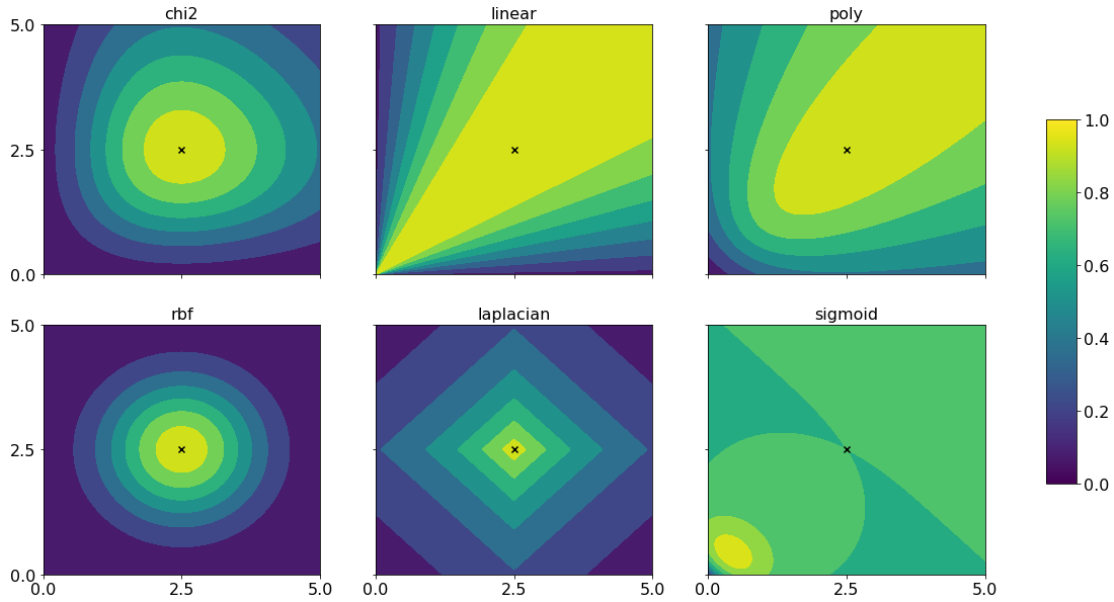


Figure 2.6.: Comparison of different (normalised) kernel functions that are possible candidates to quantify the similarity between two points. A similarity of one corresponds to perfectly similar while a similarity of zero corresponds to the two points being dissimilar. For reference, one point is fixed to the cross in the center.

Algorithm 6 Density weighted uncertainty sampling

Input: Classifier $\mathcal{F}$, unlabeled samples $\mathcal{U}$, labeled samples $\mathcal{L}$

Parameters: $\mathcal{S} \in \{\text{least confident}, \text{gini}, \text{entropy}\}$
kernel function $\mathcal{K} \in \{\text{chi2}, \text{linear}, \text{poly}, \text{rbf}, \text{laplacian}, \text{sigmoid}\}$

Output: $\boldsymbol{x}_s \in \mathcal{U}$

1: set number of centroids K to the number of unique labels Z
2: calculate K centroid locations $\mathbf{C}$ using K-Means on $\mathcal{U}$ where $\boldsymbol{C}_k$ stores a centroid location $\forall k = 1 \rightarrow K$
3: calculate $\boldsymbol{b}$ where $b_{\boldsymbol{x}_u} \in \{1, .., K\}$ is the cluster label that K-Means predicts for sample $\boldsymbol{x}_u \in \mathcal{U}$
4: compute the similarity $\boldsymbol{d}$ for all unlabeled samples to their cluster center by

$$d_{\boldsymbol{x}_u} = \frac{\mathcal{K}(\boldsymbol{x}_u, \boldsymbol{C}_{b_{\boldsymbol{x}_u}})}{\sqrt{\mathcal{K}(\boldsymbol{x}_u, \boldsymbol{x}_u)}\sqrt{\mathcal{K}(\boldsymbol{C}_{b_{\boldsymbol{x}_u}}, \boldsymbol{C}_{b_{\boldsymbol{x}_u}})}} \quad \forall \boldsymbol{x}_u \in \mathcal{U}$$

5: obtain entropy $\boldsymbol{s}$ where $s_{\boldsymbol{x}_u} = \mathcal{S}(\boldsymbol{p}_{\boldsymbol{x}_u}) \quad \forall \boldsymbol{x}_u \in \mathcal{U}$
6: $\boldsymbol{x}_s = \underset{\boldsymbol{x}_u \in \mathcal{U}}{\arg\max}\ s_{\boldsymbol{x}_u} d_{\boldsymbol{x}_u}$
7: **return** $\boldsymbol{x}_s$

classifier. Instead, CMS relies on K-Means clustering.
To make this intuitive, we assume that samples with identical labels have similar features, then different classes make up different clusters in input space. Further we assume that not only does every cluster consist of one class but also that every class has one unique cluster. Since every cluster is characterized by its cluster centroid this means a sample that is equally close (Euclidean distance) to two cluster centroids shares features with both classes. As the sample shares features with both classes it is close to the decision boundary between the two classes.
Thus we use a K-Means clustering algorithm on, for example, all available data points to find cluster centroids. Due to our assumptions the number of cluster centroids is the number of classes. Then, we query the unlabeled sample that is equally close to its two nearest centroids. By assumption, this sample is close to a decision boundary and thereby informative.
The algorithm is outlined in pseudo code in 7.

Algorithm 7 Cluster margin sampling

Input: Unlabeled samples $\mathcal{U}$, labeled samples $\mathcal{L}$
Parameters: $\mathcal{R} \in \{\mathcal{L}, \mathcal{U}, \mathcal{S}\}$
Output: $\boldsymbol{x}_s \in \mathcal{U}$

1: set number of centroids K to the number of unique labels Z
2: calculate K centroid locations using K-Means on $\mathcal{R}$
3: calculate the distance to every centroid $\boldsymbol{d}_{\boldsymbol{x}_u} \quad \forall \boldsymbol{x}_u \in \mathcal{U}$
4: $\boldsymbol{x}_s = \arg\min_{\boldsymbol{x}_u \in \mathcal{U}} (\min_2 \boldsymbol{d}_{\boldsymbol{x}_u} - \min_1 \boldsymbol{d}_{\boldsymbol{x}_u})$ # $\min_n \boldsymbol{a}$ is the n-th smallest element of $\boldsymbol{a}$
5: **return** $\boldsymbol{x}_s$

2.1.4.8. Query by committee (QbC)

QbC uses different classifiers to find informative samples. We say that samples close to the decision boundary are informative. Samples close to a decision boundary are most difficult to classify. Then, for such a sample a small change in the classifier will lead to the largest change in the classifier's label prediction. Evidently, the sample on which different classifiers deliver the most disagreement in the label prediction is the sample which is closest to a decision boundary.
Accordingly, the notion is that a committee of different classifiers predicts class labels or class membership probabilities and the sample on which the committee's prediction disagrees the most is queried.
To quantify the disagreement two measures are used - vote entropy and Kullback-Leibler divergence. To compute vote entropy the class assignments of every classifier are sufficient whereas for Kullback-Leibler divergence the class probability distributions are required.
To compute vote entropy we first gather the label prediction of all classifiers for all samples. Then, we use those predictions to estimate the class membership probability

vector of each sample. We query the sample with the highest entropy in its estimated class membership probability vector.
When using Kullback-Leibler divergence, we first gather the class membership probabilities predicted by all classifiers and for each sample. Then, for each sample we calculate the mean class membership probability vector by averaging over the classifiers. We query the sample where, on average, the classifier's predicted class membership probability vector deviates the most from the mean class membership probability vector. We use Kullback-Leibler divergence to quantify the deviation.
The query by committee sampling is outlined in algorithm 8 using vote entropy, and in algorithm 9 using Kullback-Leibler divergence.
Finally, we note that the classifiers do not necessarily have to be different if at each query iteration the training pool of every classifier is drawn with replacement independently from the available labeled data.

Digression: Understanding Kullback-Leibler divergence

Kullback-Leibler divergence is also known as relative entropy and quantifies the difference of two probability distributions. It is defined as

$$\mathcal{KL}(\mathcal{V}, \mathcal{W}) \coloneqq \sum_{x \in \mathcal{X}} \mathcal{V}(x) \log \frac{\mathcal{V}(x)}{\mathcal{W}(x)} \tag{2.8}$$

for discrete probability distributions $\mathcal{V}$ and $\mathcal{W}$ of the random variable $\mathcal{X}$.
Entropy quantifies the amount of information in a probability distribution. Let's say we have a random variable $\mathcal{X}$ that takes on the values a, b, c with probabilities $\mathcal{W}$: $1/2, 1/4, 1/4$ [31]. To trivially store the raw data of every toss we need two bits, but the entropy or information of $\mathcal{W}$ is 1.5 bits. That is because there exists an encoding that lets us store the raw data using only 1.5 bits on average. Here it is called Huffman encoding and it is given by

$$a \to 0, b \to 10 \text{ and } c \to 11. \tag{2.9}$$

Now consider a second probability distribution $\mathcal{V}$: $1/4, 1/4, 1/2$. How many bits would we need to store its raw data on average using the ideal encoding based on $\mathcal{W}$?

$$S_{\mathcal{W}} \coloneqq \frac{1}{4}1 + \frac{1}{4}2 + \frac{1}{2}2 = \sum_{x \in \mathcal{X}} -\mathcal{V}(x) \log_2 \mathcal{W}(x) = 1.75 \tag{2.10}$$

Kullback-Leibler divergence gives the number of bits wasted when encoding $\mathcal{V}$ using the ideal encoding of $\mathcal{W}$ compared to its ideal encoding:

$$\begin{aligned} \mathcal{KL}(\mathcal{V}, \mathcal{W}) &= S_{\mathcal{W}} - \mathcal{S}(\mathcal{V}) \\ &= \sum_{x \in \mathcal{X}} -\mathcal{V}(x) \log_2 \mathcal{W}(x) - \sum_{x \in \mathcal{X}} -\mathcal{V}(x) \log_2 \mathcal{V}(x) \\ &= 1.75 - 1.5 = 0.25 \text{ (bits)} \end{aligned} \tag{2.11}$$

Algorithm 8 Query by committee - vote entropy

Input: Unlabeled samples $\mathcal{U}$, labeled samples $\mathcal{L}$, committee $\boldsymbol{F}$
Output: $\boldsymbol{x}_s \in \mathcal{U}$

1: let N_F be the number of committee members
2: train $\boldsymbol{F}$ on $\mathcal{L}$
3: obtain the committee votes $\tilde{\mathbf{V}}$ where $\tilde{V}_{n_F,\boldsymbol{x}_u,z}$ is either 0 or 1 if committee member n_F predicts class label z for sample $\boldsymbol{x}_u \quad \forall \boldsymbol{x}_u \in \mathcal{U}, n_F = 1 \to N_F, z \in \mathcal{Z}$
4: sum up the member votes $V_{\boldsymbol{x}_u,z} = \sum_{n_F=1}^{N_F} \tilde{V}_{n_F,\boldsymbol{x}_u,z}$
5: turn into probabilities $\boldsymbol{p'}_{\boldsymbol{x}_u} = \frac{\boldsymbol{V}_{\boldsymbol{x}_u}}{N_F}$
6: $\boldsymbol{x}_s = \arg\max_{\boldsymbol{x}_u \in \mathcal{U}} \mathcal{S}(\boldsymbol{p'}_{\boldsymbol{x}_u})$
7: **return** $\boldsymbol{x}_s$

Algorithm 9 Query by committee - Kullback-Leibler divergence

Input: Unlabeled samples $\mathcal{U}$, labeled samples $\mathcal{L}$, committee $\boldsymbol{F}$
Output: $\boldsymbol{x}_s \in \mathcal{U}$

1: set N_F as number of committee members
2: train $\boldsymbol{F}$ on $\mathcal{L}$
3: obtain the committee votes $\tilde{\mathbf{V}}$ where $\tilde{V}_{n_F,\boldsymbol{x}_u,z}$ is the of committee member n_F predicted probability that sample $\boldsymbol{x}_u$ has label $z \quad \forall \boldsymbol{x}_u \in \mathcal{U}, n_F = 1 \to N_F, z \in \mathcal{Z}$
4: calculate mean distribution consensus $C_{\boldsymbol{x}_u,z} = \frac{1}{N_F} \sum_{n_F=1}^{N_F} \tilde{V}_{n_F,\boldsymbol{x}_u,z}$
5: compute $\mathbf{K}$ where

$$K_{n_F,\boldsymbol{x}_u} = \sum_{z \in \mathcal{Z}} \tilde{V}_{n_F,\boldsymbol{x}_u,z} \log \frac{\tilde{V}_{n_F,\boldsymbol{x}_u,z}}{C_{\boldsymbol{x}_u,z}}$$

6: take the average $K_{\boldsymbol{x}_u} \leftarrow \frac{1}{N_F} \sum_{n_F=1}^{N_F} K_{n_F,\boldsymbol{x}_u}$
7: $\boldsymbol{x}_s = \arg\max_{\boldsymbol{x}_u \in \mathcal{U}} K_{\boldsymbol{x}_u}$
8: **return** $\boldsymbol{x}_s$

2.1.4.9. Fisher information sampling (FIS)

FIS is designed to query informative samples. It defines an informative sample as a sample that leads to a high certainty in the model's (classifier's) parameters.
The inverse Fisher information $\mathbf{Q}^{-1}$ (see eq. (2.12)) lower bounds the covariance matrix of the estimated model parameter $\boldsymbol{\theta}$. Consequently, the product of the variances corresponding to the individual model parameters is lower bound by $(\det(\mathbf{Q}))^{-1}$ [18].
Then, the query strategy chooses the samples that lead to a *large* determinant in order to reduce the variance or uncertainty of the model parameters. The Fisher information matrix is given by

$$\mathbf{Q} = \mathbb{E}_{\boldsymbol{x},y}\left[\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\right)\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\right)^T\right] \approx \frac{1}{L}\sum_{\boldsymbol{x}_l \in \mathcal{L}} \mathbb{E}_y\left[\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\right)\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\right)^T\right] \propto \sum_{\boldsymbol{x}_l \in \mathcal{L}} \mathbb{E}_y\left[\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\right)\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\right)^T\right] \tag{2.12}$$

where we can use probabilities provided by a classifier to compute the expectation over the label assignment. Then, the active selection becomes

$$\boldsymbol{x}_s = \underset{\boldsymbol{x}_u \in \mathcal{U}}{\arg\max}\ \det\left\{\sum_{\boldsymbol{x}_l \in (\mathcal{L} \cup \{\boldsymbol{x}_u\})} \mathbb{E}_y\left[\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\right)\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}\right)^T\right]\right\}. \tag{2.13}$$

Computing this quantity can be, depending on the loss-function, highly non-trivial, so let's make this more explicit and continue with logistic regression and binary classification ($y \in \{1, -1\}$) as done in the paper [18]. The loss of logistic regression is given by, we will talk about this in more detail in section 3.1,

$$\mathcal{L} \coloneqq -\ln \mathcal{J}(\boldsymbol{x}, y, \boldsymbol{w}) = -\ln \sigma(y \boldsymbol{w}^T \boldsymbol{x}) \tag{2.14}$$

where we have padded the input $\boldsymbol{x} \to \boldsymbol{x} = (1, x_1, x_2, ..)^T$ to account for the intercept or bias term in $\boldsymbol{w}$ and $\sigma(x) = \frac{1}{1+\exp(-x)}$. Plugging eq. (2.14) into eq. (2.12)(with $\boldsymbol{\theta} \to \boldsymbol{w}$ and $\mathcal{L} = -\ln \mathcal{J}$) leads to

$$\begin{aligned}
Q_{i,j} &\approx \sum_{\boldsymbol{x}_l \in \mathcal{L}} \mathbb{E}_y\left[\frac{\partial \ln \mathcal{J}}{\partial w_i}\frac{\partial \ln \mathcal{J}}{\partial w_j}\right] && (2.15)\\
&= \sum_{\boldsymbol{x}_l \in \mathcal{L}} \mathbb{E}_y\left[\sigma(-y\boldsymbol{w}^T\boldsymbol{x}_l)^2 y^2 x_i x_j\right] && (2.16)\\
&= \sum_{\boldsymbol{x}_l \in \mathcal{L}} \mathbb{E}_y\left[\sigma(-y\boldsymbol{w}^T\boldsymbol{x}_l)^2 x_i x_j\right] && (2.17)\\
&= \sum_{\boldsymbol{x}_l \in \mathcal{L}} \left(\sigma(\boldsymbol{w}^T\boldsymbol{x}_l)\sigma(-\boldsymbol{w}^T\boldsymbol{x}_l)^2 x_i x_j + \sigma(-\boldsymbol{w}^T\boldsymbol{x}_l)\sigma(+\boldsymbol{w}^T\boldsymbol{x}_l)^2 x_i x_j\right) && (2.18)\\
&= \sum_{\boldsymbol{x}_l \in \mathcal{L}} \left(\sigma(\boldsymbol{w}^T\boldsymbol{x}_l) + \sigma(-\boldsymbol{w}^T\boldsymbol{x}_l)\right)\sigma(+\boldsymbol{w}^T\boldsymbol{x}_l)\sigma(-\boldsymbol{w}^T\boldsymbol{x}_l) x_i x_j && (2.19)\\
&= \sum_{\boldsymbol{x}_l \in \mathcal{L}} \sigma(+\boldsymbol{w}^T\boldsymbol{x}_l)\sigma(-\boldsymbol{w}^T\boldsymbol{x}_l) x_i x_j && (2.20)\\
&= \sum_{\boldsymbol{x}_l \in \mathcal{L}} \sigma(\boldsymbol{w}^T\boldsymbol{x}_l)(1 - \sigma(\boldsymbol{w}^T\boldsymbol{x}_l)) x_i x_j && (2.21)
\end{aligned}$$

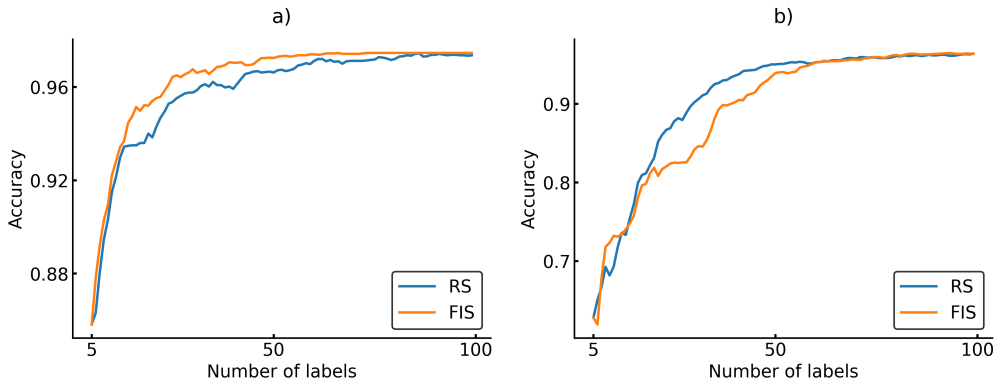


Figure 2.7.: The figure displays the classifier accuracy on test data as a function of the number of labels. It shows that eq. (2.23) has inferior performance when using a SVM with radial basis function kernel b) as opposed to a linear kernel function a). Dataset is Iris and averaged over 100 runs.

where we use from eq. (2.16) to (2.17) that $y \in \{1, -1\} \rightarrow y^2 = 1$, and from eq.(2.17) to (2.18) that $\mathcal{P}(y|\boldsymbol{x}_l, \boldsymbol{w}) = \sigma(y\boldsymbol{w}^T\boldsymbol{x}_l)$.
And then the active selection in eq. (2.13) for logistic regression is

$$\boldsymbol{x}_s = \underset{\boldsymbol{x}_u \in \mathcal{U}}{\arg\max}\ \det\left\{\mathbf{Q} + \sigma(\boldsymbol{w}^T\boldsymbol{x}_u)(1 - \sigma(\boldsymbol{w}^T\boldsymbol{x}_u))\boldsymbol{x}_u\boldsymbol{x}_u^T\right\}. \tag{2.22}$$

This explicit form for the active selection obviously is only valid for (binary) logistic regression. From analogy we can update the expression (2.22) to apply to non-binary classification leading to

$$\boldsymbol{x}_s = \underset{\boldsymbol{x}_u \in \mathcal{U}}{\arg\max}\ \det\left\{\mathbf{Q} + \mathit{Gini}(\boldsymbol{p}_{\boldsymbol{x}_u})\boldsymbol{x}_u\boldsymbol{x}_u^T\right\} \tag{2.23}$$

where $\boldsymbol{p}_{\boldsymbol{x}_u}$ is the class probability mass function (see eq. (2.1)), and the Gini-function was defined in chapter 2.1.4.2. It is straightforward to show that replacing $\boldsymbol{w}^T\boldsymbol{x} \rightarrow \mathcal{K}(\boldsymbol{w}, \boldsymbol{x})$ by an arbitrary kernel-function $\mathcal{K}$ leads to

$$\boldsymbol{x}_s = \underset{\boldsymbol{x}_u \in \mathcal{U}}{\arg\max}\ \det\left\{\mathbf{Q} + \mathit{Gini}(\boldsymbol{p}_{\boldsymbol{x}_u})(\nabla_{\boldsymbol{w}}\mathcal{K}(\boldsymbol{w}, \boldsymbol{x}_u))(\nabla_{\boldsymbol{w}}\mathcal{K}(\boldsymbol{w}, \boldsymbol{x}_u))^T\right\}. \tag{2.24}$$

Unfortunately calculating the quantity in eq. (2.13) for more complex classifiers quickly becomes intractable.
The implementation in code on github (see chapter 2.1.6) uses eq. (2.23) resulting in subpar performance for classifiers with non-linear kernels. This can be seen in figure 2.7. It should be noted that FIS was (almost) always outperformed by simpler strategies such as US and thus, did not prove useful.

2.1.4.10. Class balance sampling (CBS)

CBS neither queries informative nor representative samples, instead its idea is that a classifier performs better if the samples used for training are balanced w.r.t. class

membership, i.e., if in the training data the number of samples per class is equal for all classes.
At each training iteration the query strategy estimates the following quantity for each unlabeled sample $\boldsymbol{x}_u \in \mathcal{U}$

$$\mathcal{U}(\boldsymbol{x}_u) \coloneqq \sum_{z \in \mathcal{Z}} \left(1 - \frac{N_z}{L}\right) p_{\boldsymbol{x}_u, z} \tag{2.25}$$

with N_z the number of samples with label $z \in \mathcal{Z}$, L the total number of labeled samples and $\boldsymbol{p}_{\boldsymbol{x}_u}$ the class probabilities of sample $\boldsymbol{x}_u$. We refer to $\frac{N_z}{L}$ as the class ratios. The sample *maximizing* $\mathcal{U}$ is chosen to be queried next.
Figure 2.8 displays the quantity $\mathcal{U}$ in the case of binary classification, i.e., the class probabilities and class ratios fulfill $p_{\boldsymbol{x},1} + p_{\boldsymbol{x},2} = 1 \quad \forall \boldsymbol{x}$ and $\frac{N_1}{L} + \frac{N_2}{L} = 1$.

As an exception we look at an experimental plot immediately. Figure 2.9 compares class balance sampling to random sampling using the Iris dataset and a linear SVM. The plot is averaged over 1000 runs. We can nicely see the query strategy working as expected in the sense that it balances the training pool during the training process. The problem is that the core assumption seems to be flawed since the classifier trained on the more balanced training data performs worse.
While this plot is one of the more dramatic examples, the general scheme of class balance sampling not performing well continues throughout most datasets. We conclude that it might be useful as part of a committee of query strategies but not stand-alone.

2.1.4.11. Expected error reduction (EER)

EER, again, neither queries informative nor representative samples but is aimed to directly increase the classifier's performance. Therefore, it is referred to as a performance-based model as opposed to based on informative- or representativeness [1].
The idea is to add an unlabeled sample with every possible class label to the labeled pool and then, retrain the classifier. Next, calculate the uncertainty of every sample in the remaining unlabeled pool. We choose the sample in the query that *minimizes* the expected uncertainty in the predicted class membership probability vector of all remaining, unlabeled samples. The expectation over the label distribution of the chosen sample is done using the predicted class membership probability vector of the classifier. In that sense, we choose the sample that is the most likely to give us the highest certainty in the label prediction of the remaining samples in the unlabeled pool, and consequently, have the highest probability to correctly predict their respective labels.
We start by extending the notation of predicted class probabilities to include the option of varying the training samples used to fit the classifier that predicts the class probabilities. This leads to

$$\boldsymbol{p}_{\boldsymbol{x}_i}(\mathcal{A}) \coloneqq \mathcal{P}(\mathcal{F}_{\mathcal{A}}, \boldsymbol{x}_i) \tag{2.26}$$

where $\mathcal{F}_{\mathcal{A}}$ refers to the classifier trained on the (labeled) samples in $\mathcal{A}$.
So, $p_{\boldsymbol{x}_i, z}(\mathcal{L})$ is the predicted probability, computed by the classifier $\mathcal{F}$ trained on $\mathcal{L}$, that

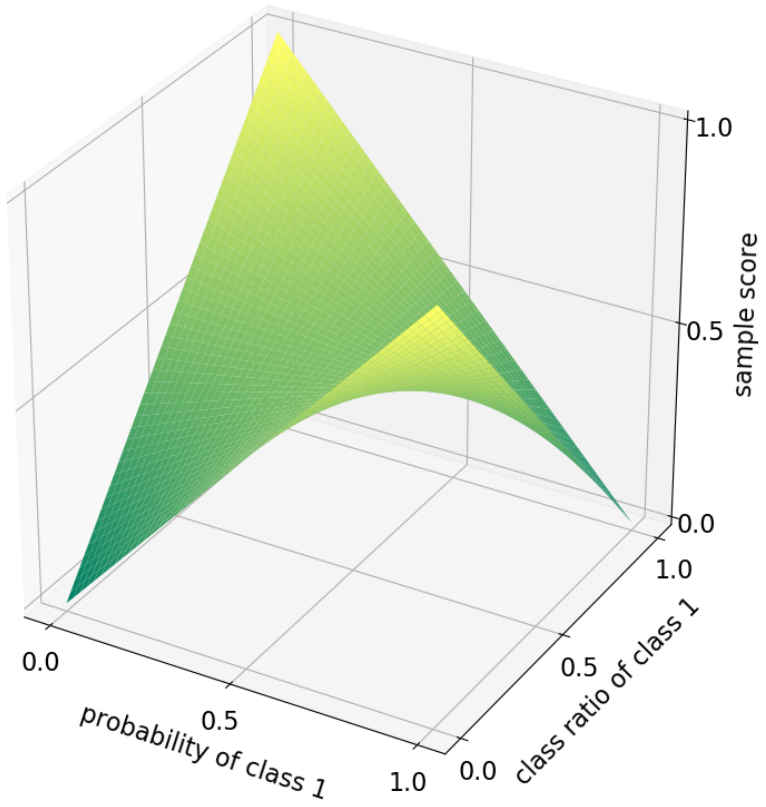


Figure 2.8.: Plot of the utility score $\mathcal{U}$, see eq. (2.25), as a function of the class probability and class ratio in the case of binary classification. Remember that it then holds $p_{\boldsymbol{x},1} + p_{\boldsymbol{x},2} = 1 \quad \forall \boldsymbol{x}$ and $\frac{N_1}{L} + \frac{N_2}{L} = 1$. Class balance sampling chooses the sample to be queried next that *maximizes* $\mathcal{U}$, since the higher the $\mathcal{U}$ the more likely the sample is to balance the labeled pool.

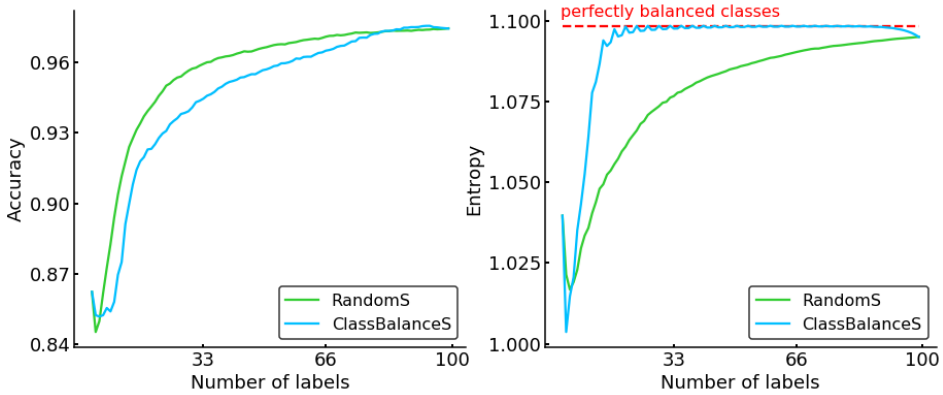


Figure 2.9.: Comparing class balance sampling and random sampling using a linear SVM on the Iris dataset. The left plot shows class balance sampling performing worse compared to random sampling, despite it working properly as indicated by the right plot. The y-axis in the right plot is to be understood as the entropy of class ratios. Averaged over 1000 runs.

sample $\boldsymbol{x}_i$ has the class label $z \in \mathcal{Z}$. This allows us to write expected error reduction as

$$\boldsymbol{x}_s = \underset{\boldsymbol{x}_u \in \mathcal{U}}{\arg\min} \left(\sum_{z \in \mathcal{Z}} p_{\boldsymbol{x}_u, z}(\mathcal{L}) \sum_{\boldsymbol{x}_{u'} \in (\mathcal{U} \setminus \boldsymbol{x}_u)} \mathcal{S}\left(\boldsymbol{p}_{\boldsymbol{x}_{u'}}\left(\mathcal{L} \cup \{(\boldsymbol{x}_u, z)\} \right) \right) \right) \tag{2.27}$$

with $\mathcal{S}$ an uncertainty measure that increases as uncertainty increases, e.g., entropy. EER is summarized in algorithm 10.

Algorithm 10 Expected error reduction

Input: Classifier $\mathcal{F}$, unlabeled samples $\mathcal{U}$, labeled samples $\mathcal{L}$
Parameters: $\mathcal{S} \in \{\text{least confident}, \text{gini}, \text{entropy}\}$
Output: $\boldsymbol{x}_s \in \mathcal{U}$

train $\mathcal{F}$ on $\mathcal{L}$
calculate class probabilities $\boldsymbol{p}_{\boldsymbol{x}_u} \quad \forall \boldsymbol{x}_u \in \mathcal{U}$
for $\boldsymbol{x}_u$ in $\mathcal{U}$ **do**
  for z in $\mathcal{Z}$ **do**
    train $\mathcal{F}$ on $\mathcal{L} \cup \{(\boldsymbol{x}_u, z)\}$
    calculate posterior class probabilities $\tilde{\boldsymbol{p}}_{\boldsymbol{x}_{u'}} \quad \forall \boldsymbol{x}_{u'} \in (\mathcal{U} \setminus \boldsymbol{x}_u)$
    sum up the posterior uncertainty of all remaining unlabeled samples

$$\tilde{s}_{z,\boldsymbol{x}_u} = \sum_{\boldsymbol{x}_{u'} \in (\mathcal{U} \setminus \boldsymbol{x}_u)} \mathcal{S}(\tilde{\boldsymbol{p}}_{\boldsymbol{x}_{u'}})$$

  end for
  compute the expected uncertainty $s_{\boldsymbol{x}_u} = \boldsymbol{p}_{\boldsymbol{x}_u} \tilde{\boldsymbol{s}}_{\boldsymbol{x}_u}$
end for
$\boldsymbol{x}_s = \arg\min_{\boldsymbol{x}_u \in \mathcal{U}} s_{\boldsymbol{x}_u}$
return $\boldsymbol{x}_s$

2.1.5. Combining query strategies

We may want to choose a sample based on several criteria. E.g., we may want to query an informative and representative sample. In fact in most cases, it is beneficial to use more than one criterion/query strategy to decide which samples to label. This section is designated to describing methods that combine an arbitrary number of the query strategies from the previous section. We refer to the query strategies of the previous section as base query strategies. This modular approach allows us to mix query strategies together freely. Doing so helps us to overcome the previously mentioned sampling bias and will reduce variance, as we will see later, and thus lead to a more robust query strategy.
First, we will introduce a trivial way (RankS) of combining multiple base query strategies which weighs every query strategy always equal. Then, we discuss two more query strategies (ALBL and DEAL) that both combine multiple base query strategies but, in contrast to RankS, are able to automatically put more emphasis on the, seemingly, more important base query strategies.

2.1.5.1. Rank sampling (RankS)

The most straightforward way of combining base query strategies. RankS chooses a sample that performs well in multiple query strategies. Every base query strategy computes the ranks of all unlabeled samples. The ranking of a query strategy goes from one ,for the best performing, to the number of unlabeled samples. E.g., consider an unlabeled pool with five samples, so $U = 5$. Then, the rank of the sample that the query strategy proposes first, is one. After removing the sample with rank one, the rank of the sample that the query strategy proposes then, is two. We continue till there is only one sample left in the unlabeled pool, and the rank of this sample is five.
The sample with the best combined ranking is queried. The summation of different hierarchies is performed in a linear or exponential way.
Both methods are outlined in detail in algorithm 11.

2.1.5.2. Active Learning by Learning (ALBL)

ALBL is a more advanced way of combining base query strategies. In contrast to RankS, the weighting of different query strategies is no longer constant and equal, but instead, it is allowed to dynamically change during the learning process. The goal is to induce a weight shift during learning towards the most promising query strategies.
This approach is based on the multi-armed bandit problem (MAB). Therefore, we start by introducing the MAB problem and discuss one general solution, called Thompson Sampling [26].

Algorithm 11 Rank sampling

Input: committee of query strategies $\boldsymbol{Q} = (Q_1, .., Q_{N_Q})^T$
Parameter: method $g \in \{\text{linear}, \text{exponential}\}$, $\alpha = 0.1$
Output: $\boldsymbol{x}_s \in \mathcal{U}$

let N_Q be the number of query strategies in $\boldsymbol{Q}$
get ranking $\mathbf{R}$ where $R_{\boldsymbol{x}_u, n_Q}$ is the rank of sample $\boldsymbol{x}_u$ in the query of $Q_{n_Q} \quad \forall \boldsymbol{x}_u \in \mathcal{U}, n_Q = 1 \to N_Q$
if v is linear **then**
    $R_{\boldsymbol{x}_u} \leftarrow \sum_{n_Q=1}^{N_Q} R_{\boldsymbol{x}_u, n_Q}$
else if v is exponential **then**
    $R_{\boldsymbol{x}_u} \leftarrow \sum_{n_Q=1}^{N_Q} -\exp\left\{-\alpha R_{\boldsymbol{x}_u, n_Q}\right\}$
end if
$s = \arg\min_{\boldsymbol{x}_u \in \mathcal{U}} R_{\boldsymbol{x}_u}$
return $\boldsymbol{x}_s$

Digression: MAB and Thompson Sampling

A one-armed bandit is a rather famous slot machine that casinos still use to "steal" their clients money. A multi-armed bandit is a one-armed bandit with multiple levers that can be pulled. The colorful name of our problem is derived from the dilemma a player faces after he sits down at such a machine. Every arm or lever produces a random reward i.i.d. drawn from an underlying distribution. Every arm has its own, to the player unknown distribution. The goal of the player is to maximize his reward given a limited budget. The task is to find a successful scheme that properly balances exploring the different arms and exploiting the acquired knowledge over the distributions.
For better understanding consider the Bernoulli Bandit as an example.
A Bernoulli Bandit is a slot machine whose arms either produce a reward of one with a certain probability or zero otherwise. Suppose there are K arms, and every arm $k \in \{1, ..., K\}$ produces a one with probability $\theta_k \in [0, 1]$. The player chooses a beta-distributed prior belief of the form

$$\mathcal{B}eta(\theta_k, \alpha_k, \beta_k) = c_{\text{Beta}}\, \theta_k^{\alpha_k - 1} (1 - \theta_k)^{\beta_k - 1} \quad \forall k = 1 \to K, \tag{2.28}$$

where c_{Beta} is a constant necessary for normalization. He initially has no knowledge and for the prior belief to be homogeneous, he chooses $\alpha_k = \beta_k = 1$.
In the first iteration $t = 1$ the player applies an action $x_1 \in \{1, ..., K\}$ and receives a reward $r_1 \in \{0, 1\}$. Having observed the reward he updates his prior according to Bayes rule and having chosen beta-distributions, the update rules are of the simple

form

$$(\alpha_k, \beta_k) \leftarrow \begin{cases} (\alpha_k, \beta_k) & \text{if } x_t \neq k \\ (\alpha_k, \beta_k) + (r_t, 1 - r_t) & \text{if } x_t = k \end{cases}. \tag{2.29}$$

Note that only the distributions of the selected action are changing.
We compare two possible strategies of the player: Either a greedy algorithm, or Thompson Sampling.
In the greedy approach the player always chooses the action k whose distribution has the highest mean reward $\hat{\theta}_k$.
Thompson Sampling on the other hand draws randomly and uses the result as an estimate for the mean reward $\hat{\theta}_k$ from the corresponding beta-distribution and similarly frequently pulls the action with the highest mean reward.
Both algorithms are in pseudo code form in algorithm 12.

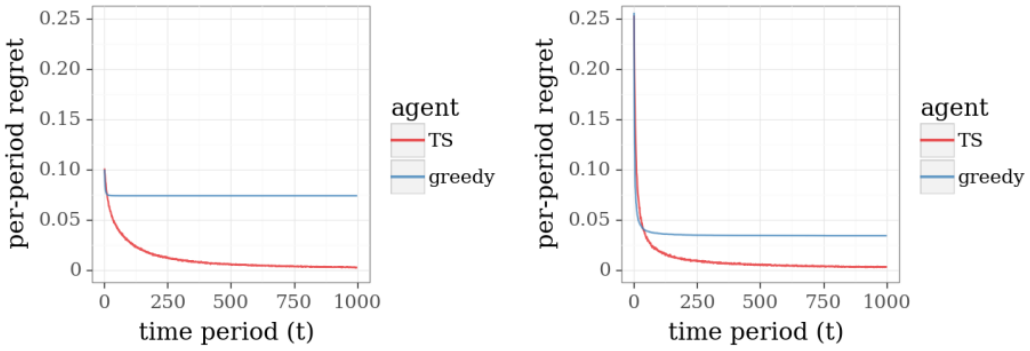
Finally, we define the per-period regret of an algorithm as the difference between the mean reward of an optimal action as opposed to the mean reward of the action picked by the algorithm. Therefore, a to zero converging regret also implies that the algorithm always "finds" the optimal action. In figure 2.10 the per-period regret is plotted versus the number of iterations for Thompson- and greedy sampling in case of a three-armed Bernoulli Bandit. It is an average of 10.000 runs.

Algorithm 12 Greedy- and Thompson Sampling

for $t = 1 \rightarrow ..$ **do**
  for $k = 1 \rightarrow K$ **do**
    if Greedy Samling **then**
      $\hat{\theta}_k = \frac{\alpha_k}{\alpha_k + \beta_k} = \overline{\mathcal{B}eta(\theta_k, \alpha_k, \beta_k)}$
    else if Thompson Sampling **then**
      draw $\hat{\theta}_k$ from $\mathcal{B}eta(\theta_k, \alpha_k, \beta_k)$
    end if
  end for
  # select and apply action
  $x_t = \arg\max_{k=1 \rightarrow K} \hat{\theta}_k$
  Apply x_t and observe r_t
  # update distribution
  $(\alpha_{x_t}, \beta_{x_t}) \leftarrow (\alpha_{x_t} + r_t, \beta_{x_t} + 1 - r_t)$
end for

Having understood Thompson Sampling, we draw a similar analogy as in [14] where the different arms (options to choose from) are query strategies. Each query strategy has a certain weight associated to it. The higher the weight the more likely it is chosen and after "pulling the arm", we label the sample recommended by that base query strategy and receive a reward. Next, we update the weights of the base query strategies that recommended that sample, proportional to the reward. Then, we repeat this process.

Figure 2.10.: Regret of Greedy- and Thompson sampling on the three-armed Bernoulli Bandit [26].



(a) $\theta = (0.9, 0.8, 0.7)$ **(b)** average over random θ

Several arduous questions immediately arise - first, given a set of weights what is the probability a certain arm is chosen? At this point, we only established that the higher the weight of an arm the more likely it is chosen.
The second question is, what is the update rule concerning the weights going into the next round? I.e., having observed the reward of the action proposed by the chosen arm (the reward of labeling the sample proposed by the chosen query strategy), leads to what weight update of the arms?
And finally, what is the reward function, i.e., how to quantify the information gain of adding a certain sample to the labeled pool?

The answer to the first and second question is contained in paper [5] in the form of algorithm Exp4.P. The answer to the latter question is given in paper [14] via "importance weighted accuracy" as an unbiased estimate of test accuracy.
Let's start be discussing the used Exp4.P algorithm, a modification to the one given in the paper [5]. The modified Exp4.P is given in algorithm 13. In the following, we outline its key steps and the modification we here introduce. The algorithm 13 starts by gathering the information which sample each arm proposes (code line 2). Next, the algorithm converts the weights of every arm into a probability with which each arm is chosen (code line 3), the higher the weight the more likely the arm is chosen. Then, given the arm probabilities it draws randomly an arm and, by implication, also a sample to label (code line 4, 5). This is a key step which incorporates Thompson Sampling since we draw randomly instead of simply choosing the arm with the highest probability/weight. At this point, we temporarily pause the subroutine Exp4.P and continue in the main algorithm of ALBL. It is given by algorithm 14 and starts off by using Exp4.P to choose which sample to label, which is exactly where we paused the subroutine. The main algorithm then receives the sample which is to be labeled and with which probability it was chosen (code line 3 or 5). Then, the sample is labeled by the oracle and afterwards,

it is added to the labeled pool (code line 9, 10). The main algorithm of ALBL then uses importance weighted accuracy to calculate the reward of adding the sample to the labeled pool (code line 12) and sends it back to the subroutine. We continue where we left off in the subroutine and use the received reward to update the weight of the arms that proposed the chosen sample. The weight increases the more, the higher the reward. There is also an exploration factor ($\hat{v}^k = 1/p^k(t)$) which ensures that even unlikely arms do not become completely irrelevant over time. Finally, Exp4.P can restart from the beginning with updated arm weights and propose the next sample to label.

Now, the only missing piece is the reward function. The ideal reward would be a test accuracy, but since the goal is to minimize the labels required to reach a certain accuracy, it would be counter-intuitive to waste labels on a separate test pool. To elude this problem, we use an unbiased estimate of the test accuracy called importance weighted accuracy. It is defined as

$$I(\mathcal{F}, \tau) \coloneqq \frac{1}{NT} \sum_{t=1}^{\tau} \omega_t [\![y_{u_t} = \mathcal{F}(\boldsymbol{x}_{u_t})]\!] \tag{2.30}$$

where $\omega_t = (q_{\boldsymbol{x}_{u_t}}(t))^{-1}$, N is the total number of samples and T is the label budget. The $[\![]\!]$-notation is defined as

$$[\![A = B]\!] = \begin{cases} 1 & \text{if } A \text{ is equal to } B, \\ 0 & \text{else.} \end{cases} \tag{2.31}$$

The key trick of importance weighted accuracy is weighing each summand in eq. (2.30) by the inverse of the probability that the sample was chosen. To give this some intuition we may think of the following - samples with a high probability of being chosen, are chosen more frequently and hence, occur often in the training data and the classifier is well fitted to them whereas unlikely samples are rare and are basically test data, so they should contribute more when estimating a test accuracy.

Finally, we note the key distinction between the here discussed approach and the approach as outlined in paper [14]. We start by quoting a paragraph of the paper:
"Without loss of generality, assuming that the k-th algorithm a_k is chosen by EXP4.P in ALBL. Then, the query request of a_k should be followed to query from D_u. To accommodate the possibility that a_k would want to make a probabilistic query, we introduce the query vector $\boldsymbol{\Psi}^k(t) \in [0,1]^{n_u}$, where its j-th component $\Psi_j^k(t)$ indicates the preference of the k-th algorithm on querying the label of $x_j \in D_u$ in iteration t. *The query vector should represent a probability distribution* to querying from D_u; that is, $\sum_{j=1}^{n_u} \Psi_j^k(t) = 1$. Deterministic active learning algorithms could simply return a degenerate query vector that contains a single 1 on its most preferred instance, and 0 elsewhere."[14]
ALBL expects query strategies to return a probability distribution that expresses the query strategies preference to query every possible sample. However, all base query strategies work by computing an objective function for every possible sample and query the sample that performs best w.r.t that objective function. The distribution of a query

strategy's objective function differs between query strategies. Thereby, converting the objective function's score distribution into a probability distribution in a comparable way is non-trivial. Which is the reason the implementation of ALBL as discussed here, treats every query strategy as a deterministic query strategy, i.e., every query strategy returns a probability distribution that is non-zero only for the query strategy's most desired sample.
This limitation is removed in the algorithm DEAL, discussed in the next chapter.

Algorithm 13 Modified Exp4.P generator for Active Learning by Learning

Parameters: $\delta > 0$ (default: $\delta = 0.1$), $p_{\min} \in [0, 1/K]$ (default: $p_{\min} = \sqrt{\frac{\ln K}{KT}}$), label budget T, number of arms K, committee of query strategies $\boldsymbol{Q} = (Q_1, .., Q_K)^T$

Initialization: Set $w^k(1) = 1 \quad \forall k = 1 \to K$

1: **for** $t = 1 \to T$ **do**

2: get advice vectors $\boldsymbol{\xi}^1(t), ..., \boldsymbol{\xi}^K(t)$ from $\boldsymbol{Q}$ where

$$\xi^k_{\boldsymbol{x}_u}(t) = \begin{cases} 1 & \text{if arm } k \text{ queries sample } \boldsymbol{x}_u \\ 0 & \text{else} \end{cases} \qquad \forall \boldsymbol{x}_u \in \mathcal{U}, k = 1 \to K$$

3: set $W_t = \sum_{k=1}^{K} w^k(t)$ and calculate

$$p^k(t) \coloneqq (1 - Kp_{\min}) \frac{w^k(t)}{W_t} + p_{\min} \quad \forall k = 1 \to K$$

4: set $q_{\boldsymbol{x}_u}(t) = \sum_{k=1}^{K} p^k(t) \xi^k_{\boldsymbol{x}_u}(t) \quad \forall \boldsymbol{x}_u \in \mathcal{U}$

5: draw sample $\boldsymbol{x}_{u_t}$ with probability $q_{\boldsymbol{x}_u}(t)$

6: receive reward r_{u_t}

7: rescale reward $r_{u_t} \leftarrow \frac{r_{u_t}}{q_{\boldsymbol{x}_{u_t}}(t)}$

8: # adjust weights of arms that voted for drawn sample $\boldsymbol{x}_{u_t}$

9: $\forall k = 1 \to K$ set

$$\hat{y}^k = \xi^k_{\boldsymbol{x}_{u_t}}(t) r_{u_t}$$

$$\hat{v}^k = \frac{1}{p^k(t)}$$

10: update weights according to

$$w^k(t+1) = w^k(t) \exp\left\{ \frac{p_{\min}}{2} \left(\hat{y}^k + \hat{v}^k \sqrt{\frac{\ln \frac{K}{\delta}}{KT}} \right) \right\} \quad \forall k = 1 \to K$$

11: **end for**

Algorithm 14 Active Learning by Learning

Input: classifier $\mathcal{F}$, labeled samples $\mathcal{L}$, unlabeled samples $\mathcal{U}$, label budget T
Output: $\boldsymbol{x}_s \in \mathcal{U}$

for $t = 1 \to T$ **do**
  if t is 1 **then**
    start modified Exp4.P and obtain $\boldsymbol{x}_{u_t} \in \mathcal{U}$ and $q_{\boldsymbol{x}_{u_t}}(t)$
  else
    obtain $\boldsymbol{x}_{u_t} \in \mathcal{U}$ and $q_{\boldsymbol{x}_{u_t}}(t)$
  end if
  store $\omega_t = (q_{\boldsymbol{x}_{u_t}}(t))^{-1}$
  return $\boldsymbol{x}_s = \boldsymbol{x}_{u_t}$
  wait for Oracle, $y_{u_t} = O(\boldsymbol{x}_{u_t})$
  move $(\boldsymbol{x}_{u_t}, y_{u_t})$ from $\mathcal{U}$ to $\mathcal{L}$
  train $\mathcal{F}$ on $\mathcal{L}$
  calculate reward $r = I(\mathcal{F}, t)$
  send reward r back to modified Exp4.P
end for

2.1.5.3. Dynamic Ensemble Active Learning (DEAL)

DEAL combines multiple base query strategies. DEAL generalises ALBL in the sense that it, in contrast to ALBL, treats every query strategy as making a probabilistic query, i.e., every query strategy proposes every possible sample with some probability.
This also implies that in DEAL the analogy shifts from a MAB problem to a MAB problem with expert advice [23]. The N_Q experts correspond to the ensemble of query strategies and the K arms are unlabeled samples in the pool. Active learners based on MAB with expert advice, e.g., [23] and [3] aim to learn the expert that recommends the arm that leads to the highest reward. This has the crucial advantage that at every iteration the weight of every query strategy can be updated since no matter which sample is labeled, every query strategy proposed that sample with some probability and is able to receive a share of the reward.

In the previous section we discussed the necessary restriction to interpreting every query strategy as deterministic in ALBL. DEAL lifts this restriction by introducing a way of converting a ranking hierarchy into a probability distribution via "exponential ranking normalization". This leads to the modification of Exp4.P (algorithm 13) leading to RExp4 in algorithm 15. There, we use exponential ranking normalisation (code line 4) and apply a Gibbs measure (code line 6) to convert the rankings of query strategies into probability distributions.
We also drop the exploration term of Exp4.P in favor of a periodic weight reset (code line $14-17$) which serves a similar purpose.
We note that, the crucial part of multiple experts receiving a reward at a single iteration is hidden in the pseudo-code. This happens in algorithm 15 in code line 13 which is similar to the corresponding part in the code of Exp4.P (there: code line 9). This was done on purpose to make the similarity between the two more clear. However, in Exp4.P $\boldsymbol{\xi}$ is a perfectly sparse vector with only one one at the location of the preferred sample of the query strategy. In RExp4 this same vector (after normalisation/after code line 8) represents a probability distribution with non-zero entropy. In fact, the amount of entropy can be adjusted with the hyperparameters α, β.
Finally, we modify importance weighted accuracy as defined in eq. (2.30) leading to

$$I(\mathcal{F}, \tau) \coloneqq \frac{\gamma}{T} \sum_{t=1}^{\tau} \frac{\omega_t}{K_t} [\![y_{u_t} = \mathcal{F}(\boldsymbol{x}_{u_t})]\!] \tag{2.32}$$

where K_t is the number of arms at iteration step t, $\gamma \in (0, 1]$ is the learning rate of RExp4, $\omega_t = (p_{\boldsymbol{x}_{u_t}}(t))^{-1}$ and T is the label budget. The difference between the two equations is the factor $1/K_t$ and it ensures that eq. (2.32) is bounded from above by one. Too see this, consider the following: Due to code line 8 in RExp4, it holds that

$$p_{\boldsymbol{x}_u}(t) = (1-\gamma) \sum_{n_Q=1}^{N_Q} \frac{w^{n_Q}(t) \xi_{\boldsymbol{x}_u}^{n_Q}(t)}{W_t} + \frac{\gamma}{K_t} \geq \frac{\gamma}{K_t} \iff \frac{\gamma \omega_t}{K_t} \leq 1 \quad \forall t \leq T, \boldsymbol{x}_u \in \mathcal{U}. \tag{2.33}$$

Thus, I in eq. (2.32) is less than one for all $\tau \leq T$.

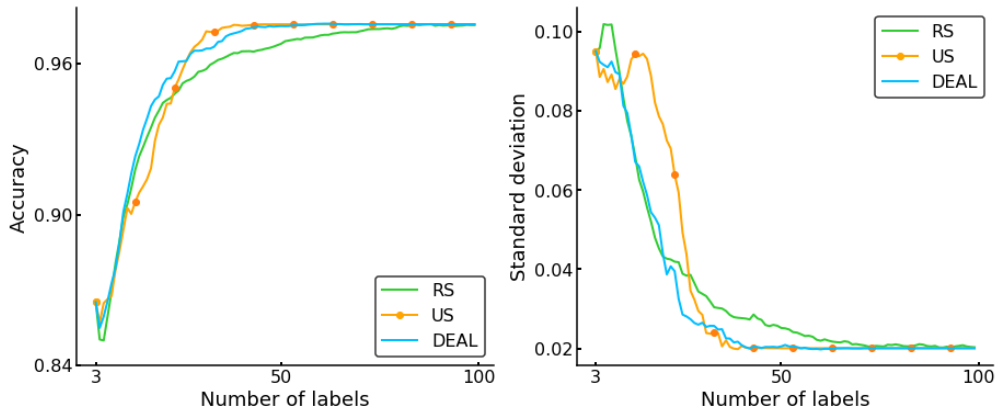


Figure 2.11.: Plot of the test accuracy and standard deviation as a function of the number of labels for a SVM with linear kernel on the Iris dataset. DEAL combines RS and US as base query strategies. In the right figure we can observe DEAL having lower standard deviation while performing comparably to US as seen in the left figure. Averaged over 500 runs.

Then, DEAL is identical to ALBL, see algorithm 14, when substituting Exp4.P for RExp4 and using the eq. (2.32) instead of eq. 2.30.

Remark: Variance reduction

One of the benefits of using an ensemble of query strategies via a bandit algorithm such as ALBL and DEAL is hidden in the later appearing accuracy plots in chapter 4 - it is variance reduction. As an example, US often improves the performance gain per label, but it does so only on average. While that is an important factor, it also means that US occasionally (or even rarely) performs worse than RS. However, when it does, it scores exceptionally poor leading to a high variance.
Giving the algorithm an alternative option when the most effective query strategy fails greatly decreases variance, making the query strategy more reliable at almost no cost. This can be seen in figure 2.11.

Algorithm 15 REXP4 generator for Dynamic Ensemble Active Learning

Parameters: learning rate $\gamma \in (0, 1]$ (default: $\gamma = 0.1$), $\Delta_T = 10$, $\alpha = 0.1$, $\beta = 10$, committee of query strategies $\boldsymbol{Q} = (Q_1, .., Q_{N_Q})^T$, label budget T, number of experts N_Q

Initialization: Set $w^{n_Q}(1) = 1 \quad \forall n_Q = 1 \to N_Q$ and $\tau = 1$

for $t = 1 \to T$ **do**
  set number of arms to number of unlabeled samples, so $K_t = U$
  get ranking vectors $\boldsymbol{\xi}^1(t), ..., \boldsymbol{\xi}^{N_Q}(t)$ from $\boldsymbol{Q}$ where $\xi^{n_Q}_{\boldsymbol{x}_u}(t)$ is the rank of sample $\boldsymbol{x}_u$ in the query of expert $n_Q \quad \forall \boldsymbol{x}_u \in \mathcal{U}, n_Q = 1 \to N_Q$
  # apply exponential ranking normalization
  $\xi^{n_Q}_{\boldsymbol{x}_u}(t) \leftarrow -\exp\left\{-\alpha \xi^{n_Q}_{\boldsymbol{x}_u}(t)\right\} \quad \forall \boldsymbol{x}_u \in \mathcal{U}, n_Q = 1 \to N_Q$
  # apply Gibbs measure
  $\xi^{n_Q}_{\boldsymbol{x}_u}(t) \leftarrow \exp\left\{-\beta \xi^{n_Q}_{\boldsymbol{x}_u}(t)\right\} \quad \forall \boldsymbol{x}_u \in \mathcal{U}, n_Q = 1 \to N_Q$
  normalize $\xi^{n_Q}_{\boldsymbol{x}_u}(t) \leftarrow \frac{\xi^{n_Q}_{\boldsymbol{x}_u}(t)}{\sum_{\boldsymbol{x}_u \in \mathcal{U}} \xi^{n_Q}_{\boldsymbol{x}_u}(t)} \quad \forall \boldsymbol{x}_u \in \mathcal{U}, n_Q = 1 \to N_Q$
  set $W_t = \sum_{n_Q=1}^{N_Q} w^{n_Q}(t)$ and $\forall \boldsymbol{x}_u \in \mathcal{U}$ calculate

$$p_{\boldsymbol{x}_u}(t) = (1-\gamma) \sum_{n_Q=1}^{N_Q} \frac{w^{n_Q}(t) \xi^{n_Q}_{\boldsymbol{x}_u}(t)}{W_t} + \frac{\gamma}{K_t}$$

  draw sample $\boldsymbol{x}_{u_t}$ with probability $p_{\boldsymbol{x}_u}(t)$
  receive reward r_{u_t}
  rescale reward $r_{u_t} \leftarrow \frac{r_{u_t}}{p_{\boldsymbol{x}_{u_t}}(t)}$
  # adjust weights of arms that voted for drawn sample $\boldsymbol{x}_{u_t}$
  for $n_Q = 1 \to N_Q$ set

$$\hat{y}^{n_Q} = r_{u_t} \xi^{n_Q}_{\boldsymbol{x}_{u_t}}(t)$$
$$w^{n_Q}(t+1) = w^{n_Q}(t) \exp\left\{\frac{\gamma}{K_t} \hat{y}^{n_Q}\right\}$$

  $\tau \leftarrow \tau + 1$
  # weight reset mechanism
  if $\tau > \Delta_T$ **then**
    set $\tau = 1$ and $\boldsymbol{\omega}(t+1) = \mathbf{1}$
  end if
end for

2.1.6. Implementation in code

To see an implementation in python of all query strategies of section 2.1.4 and 2.1.5 you may check out my repository called "pool_based_active_learning" on Github

`https://github.com/SimiPixel/pool_based_active_learning`.

You may also refer to the package "libact: Pool-based Active Learning in Python" located at

`https://github.com/ntucllab/libact`.

This completes the discussion of pool-based active learning and we move on by discussing on-line active learning.

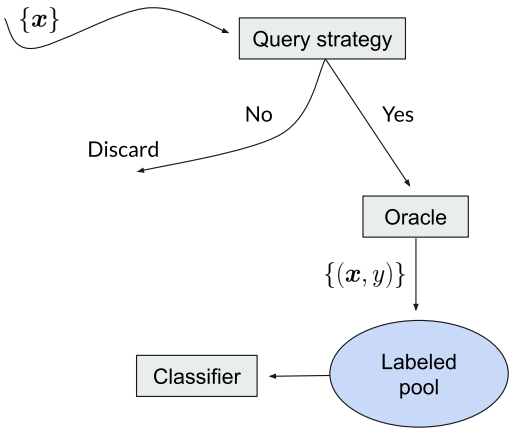


Figure 2.12: Schematic overview of on-line active learning. For every observed, unlabeled sample the query strategy decides wether or not it should receive a label. If not the sample is deleted, and otherwise it will be labeled by the oracle and added to the labeled pool. The labeled pool is then used to train a classifier.

2.2. On-line active learning

On-line active learning refers to active learning strategies where the query is shown a sample and it decides whether or not to request the sample's label. Its core motivation is to make active learning more practical as pool-based approaches often fail to be applicable in practice [19].
Some key advantages are, e.g., that it requires less storage space - the incoming sample is either labeled and added to the training pool or it is deleted, there is no need for an unlabeled sample pool. Furthermore following a simple queue strategy ("Last In - First Out") for incoming samples guarantees that the expert has to label the latest samples first. This is beneficial assuming that the expert has a higher probability of correctly labeling younger samples, i.e., the expert knows best about what happened right now, not about what happened some time in the past.
On-line active learning can also be computational less demanding since a utility score is only calculated for one sample and not an entire set of samples.

2.2.1. Components

In on-line active learning as soon as a new sample occurs, it is forwarded to the query strategy. The query strategy then decides whether or not to label the sample. If it declines, the sample is deleted. Otherwise the sample is send to the oracle for labeling. Afterwards the labeled sample is added to the labeled pool and it is used to train a classifier.
The scheme of on-line active learning is shown in figure 2.12.
Next, we introduce a popular structure of the query strategy in case of on-line active learning.

2.2.1.1. Importance weighted active learning (IWAL)

The query strategy decides whether or not to label the sample using a biased coin flip. By adapting the coin bias the sample is more likely to be labeled if a utility score indicates a high usefulness.
This approach follows the outline of "importance weighted active learning" [4, 1] and it pursues nicely the spirit of exploration and exploitation. It reduces sampling bias, see section 2.1.3, by not strictly following a certain policy but by introducing randomness.
The approach differs from pool-based active learning described in algorithm 2 in the following sense. The query strategy in pool-based active learning chooses a sample out of $\mathcal{U}$, so it immediately needs access to all samples and hence has a sample complexity of n as opposed to the query strategy of on-line active learning where the samples are drawn randomly so its sample complexity equals the sample complexity of passive learning. The difference to passive learning is that the query will sometimes decline the labeling of samples, so its label complexity will be lower than the one of passive learning while (hopefully) maintaining the same or better performance.
The query strategy of on-line active learning is described in algorithm 16 where threshold δ is some constant. It is important to note that the coin bias should be between zero and one. In algorithm 16 it is rewritten in a way that is consistent with pool-based active learning.

Algorithm 16 Query strategy of on-line active learning

Input: Classifier $\mathcal{F}$ trained on $\mathcal{L}$, unlabeled samples $\mathcal{U}$, threshold δ
Output: $\boldsymbol{x}_u \in \mathcal{U}$

for $t = 1 \rightarrow ..$ **do**
  choose $\boldsymbol{x}_{u_t} \in \mathcal{U}$ randomly from $\mathcal{U}$
  determine coin bias $b = \mathcal{B}(\boldsymbol{x}_{u_t})$ # $\mathcal{B}$ is discussed in section 2.2.2
  $b \leftarrow b + \delta$ # add threshold
  $c = \mathcal{R}andom(0, 1)$ # random real number between zero and one
  if $c \leq b$ **then**
    break # leave for loop
  end if
end for
return $\boldsymbol{x}_{u_t}(, b)$ # we can return bias to use as sample weight

2.2.1.2. Weights

Up to this point we have not yet talked about sample weights but generally we could also assign every labeled sample a weight ω_l to respect when training the classifier so

$$(\boldsymbol{x}_l, y_l) \rightarrow (\boldsymbol{x}_l, y_l, \omega_l) \quad \forall l = 1 \rightarrow L. \tag{2.34}$$

In the IWAL algorithm specified in [4] these weights are given by $\omega_{\boldsymbol{x}_{u_t}} = 1/b$ where b is the probability with which sample $\boldsymbol{x}_{u_t}$ has been drawn and labeled at iteration t in

algorithm 16.
Another possibility is to weigh each sample by

$$\omega_l = \frac{\rho_{\mathcal{S}}(\boldsymbol{x}_l)}{\rho_{\mathcal{L}}(\boldsymbol{x}_l)} \quad \forall l = 1 \to L. \tag{2.35}$$

This is well defined since $\rho_{\mathcal{L}}(\boldsymbol{x}_l) > 0 \quad \forall \boldsymbol{x}_l \in \mathcal{L}$.

2.2.2. Query strategies

At this point we have established the basic structure of the query strategy in on-line active learning: Every incoming sample gets assigned the probability of labeling the sample, then we randomly draw whether or not the sample is labeled. In this section we discuss in detail how to assign a sample its probability, whether it is labeled or not.
This corresponds to discussing the function $\mathcal{B}$ of algorithm 16.

2.2.2.1. Heterogeneity/Uncertainty sampling (US)

US makes it more likely that informative samples are labeled. As in section 2.1.4.2, we define informative samples as samples close to a decision boundary. These samples have high entropy in the class membership probability vector predicted by a classifier. The following is in complete analogy to the active learner "PER-AL" described in survey [19] where they use a Bernoulli random variable with parameter $\frac{b}{b+|p_t|}$ with p_t the distance to the separating hyperplane and b a hyperparameter that adjusts the slope.

We know that entropy is bounded between zero and $\ln Z$ where Z is the number of classes. Therefore, to determine the coin bias (high bias = more likely to request label) we can use every function (ideally a monotonously increasing one) that fulfills the property

$$\mathcal{B}\colon [0, \ln Z] \to [0, 1] \text{ with } Z \text{ the number of classes.} \tag{2.36}$$

Two possible choices are plotted in figure 2.13.
The scheme of US is summarized in algorithm 17.
The coin bias can be returned and used as inverse fitting weight with the intention of reducing sampling bias.

2.2.2.2. Expected error reduction (EER)

EER chooses the sample that is the most likely to reduce the uncertainty in the label prediction of unlabeled samples.
The difficult part in the case of on-line active learning is to determine which entropy reduction corresponds to a good sample. In the pool-based approach we simply determined the probability weighted entropy reduction of the unlabeled pool when adding a specific sample. The sample that leads to the highest reduction is queried. To reuse this hierarchic sorting we introduce a small, ten sample large unlabeled gauge pool on which

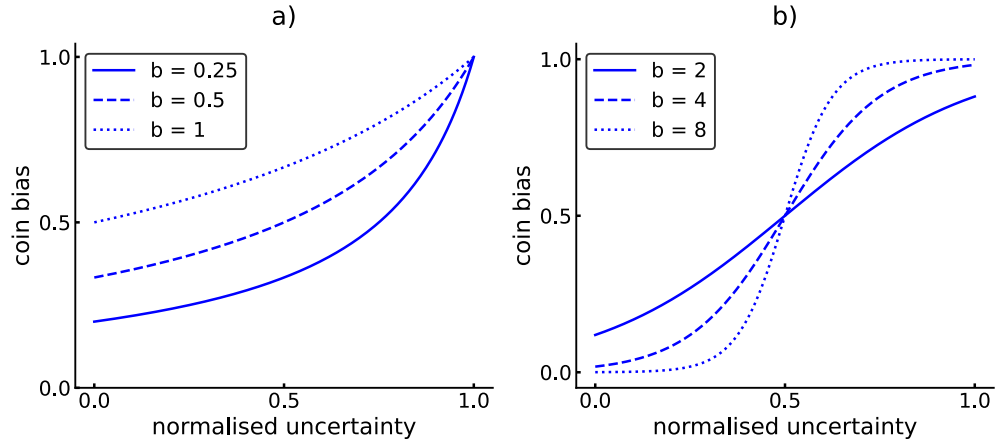


Figure 2.13.: Different functions to determine the coin bias or Bernoulli parameter of a sample given the normalised entropy of its class probabilities. Normalisation is done using the maximally possible entropy $\mathcal{S}_{\max} = \ln Z$ with Z the number of classes. The coin bias is given by $\frac{b}{b+(1-s)}$ or $\frac{1}{1+\exp\{-b(s-0.5)\}}$ where s denotes the normalised uncertainty, for a) and b) respectively.

Algorithm 17 On-line active learning using US

Input: Classifier $\mathcal{F}$ trained on $\mathcal{L}$, $\boldsymbol{x}_u \in \mathcal{U}$
Output: type string $\in \{\text{yes}, \text{no}\}$

let s be the entropy of sample $\boldsymbol{x}_u$, so $s = \mathcal{S}(\boldsymbol{p}_{\boldsymbol{x}_u})$
Normalise, $s \leftarrow \frac{s}{\ln Z}$
$p = \frac{1}{1+\exp\{-b(s-0.5)\}}$ with $b = 4$
$c = \mathit{Random}(0, 1)$ # random real number between zero and one
if $c \leq p$ **then**
    return yes, $(\frac{1}{p})$ # return $1/p$ to use as weight
else
    return no
end if

we base our decision.
Then the strategy is outlined in detail in algorithm 18 and following a weighted scheme in algorithm 19.

Algorithm 18 On-line active learning using EER

Input: Classifier $\mathcal{F}$ trained on $\mathcal{L}$, $\boldsymbol{x}_s \in \mathcal{U}$, $\mathcal{G} \subset (\mathcal{U} \setminus \{\boldsymbol{x}_s\})$
Output: type string $\in \{\text{yes}, \text{no}\}$

$\mathcal{G}' = \mathcal{G} \cup \{\boldsymbol{x}_s\}$
calculate beforehand class probabilities $\boldsymbol{p}_{\boldsymbol{x}_{g'}} \forall \boldsymbol{x}_{g'} \in \mathcal{G}'$
Z is number of classes, G number of samples in $\mathcal{G}$
declare empty array $\tilde{\mathbf{S}}$ of shape $= (Z, G' = G + 1)$# $G = 10$
for $\boldsymbol{x}_{g'}$ in $\mathcal{G}'$ **do**
  $\mathcal{G}_t \coloneqq \mathcal{G}' \setminus \{\boldsymbol{x}_{g'}\}$
  for z in $\mathcal{Z}$ **do**
    $\mathcal{L}_t = \mathcal{L} \cup \{(\boldsymbol{x}_{g'}, z)\}$ and train $\mathcal{F}$ on extended $\mathcal{L}_t$
    calculate afterwards class probabilities $\tilde{\boldsymbol{p}}_{\boldsymbol{x}_{g_t}} \forall \boldsymbol{x}_{g_t} \in \mathcal{G}_t$
    set $\tilde{S}_{z,\boldsymbol{x}_{g'}} = \sum_{\boldsymbol{x}_{g_t} \in \mathcal{G}_t} \mathcal{S}(\tilde{\boldsymbol{p}}_{\boldsymbol{x}_{g_t}})$
  end for
end for
(weighted) entropy reduction $\tilde{S}_{\boldsymbol{x}_{g'}} \leftarrow \boldsymbol{p}_{\boldsymbol{x}_{g'}} \tilde{\boldsymbol{S}}_{\boldsymbol{x}_{g'}} \quad \forall \boldsymbol{x}_{g'} \in \mathcal{G}'$
calculate mean and standard deviation μ and σ of $\tilde{\boldsymbol{S}}$
$p = \frac{1}{1+\exp\left\{-b\frac{\tilde{S}_{\boldsymbol{x}_s} - \mu}{\sigma}\right\}}$ with $b = 4$
$c = \mathit{Random}(0, 1)$ # random real number between zero and one
if $c \leq p$ **then**
  return yes
else
  return no
end if

Algorithm 19 On-line active learning using EER with weights

Input: Classifier $\mathcal{F}$ trained on $\mathcal{L}$, $\boldsymbol{x}_s \in \mathcal{U}$, $\mathcal{G} \subset (\mathcal{U} \setminus \{\boldsymbol{x}_s\})$, vector of weights $\boldsymbol{w}$
Output: type string $\in \{\text{yes}, \text{no}\}$, weight ω

if $\boldsymbol{w}$ is empty **then**
  $w_l = 2 \quad \forall l = 1 \rightarrow L+1$
end if
initialize ω as mean of $\boldsymbol{w}$
$\mathcal{G}' = \mathcal{G} \cup \{\boldsymbol{x}_s\}$
calculate beforehand class probabilities $\boldsymbol{p}_{\boldsymbol{x}_{g'}} \forall \boldsymbol{x}_{g'} \in \mathcal{G}'$
iterPoint
declare empty array $\tilde{\mathbf{S}}$ of shape $= (Z, G' = G+1)$# $G = 10$
for $\boldsymbol{x}_{g'}$ in $\mathcal{G}'$ **do**
  $\mathcal{G}_t \coloneqq \mathcal{G}' \setminus \{\boldsymbol{x}_{g'}\}$
  $w_{L+1} = \omega$
  for z in $\mathcal{Z}$ **do**
    $\mathcal{L}_t = \mathcal{L} \cup \{(\boldsymbol{x}_{g'}, z)\}$ and train $\mathcal{F}$ on extended $\mathcal{L}_t$ w.r.t weights $\boldsymbol{w}$
    calculate afterwards class probabilities $\tilde{\boldsymbol{p}}_{\boldsymbol{x}_{g_t}} \forall \boldsymbol{x}_{g_t} \in \mathcal{G}_t$
    set $\tilde{S}_{z,\boldsymbol{x}_{g'}} = \sum_{\boldsymbol{x}_{g_t} \in \mathcal{G}_t} \mathcal{S}(\tilde{\boldsymbol{p}}_{\boldsymbol{x}_{g_t}})$
  end for
end for
(weighted) entropy reduction $\tilde{S}_{\boldsymbol{x}_{g'}} \leftarrow \boldsymbol{p}_{\boldsymbol{x}_{g'}} \tilde{\boldsymbol{S}}_{\boldsymbol{x}_{g'}} \quad \forall \boldsymbol{x}_{g'} \in \mathcal{G}'$
calculate mean and standard deviation μ and σ of $\tilde{\boldsymbol{S}}$
$p = \frac{1}{1+\exp\left\{-b\frac{\tilde{S}_{\boldsymbol{x}_s} - \mu}{\sigma}\right\}}$ with $b = 4$
$\omega = 1/p$
if $|\omega - w_{l+1}| > \epsilon$ **then**
  goto iterPoint # e.g. $\epsilon = 0.01$
end if
$c = \mathcal{R}\mathit{andom}(0,1)$ # random real number between zero and one
if $c \leq p$ **then**
  return yes, ω # return weight
else
  return no
end if

3. Auxiliary data source

As already mentioned, the ulterior motive is to label efficiently when obtaining the training data necessary for a supervised classification. But since recruiting a human expert as oracle is expensive, it may be a valid option to use some non-human entity as oracle instead. This results in a cheaply obtainable and possibly large training dataset with sub optimal label accuracy. The goal is to exploit such an auxiliary pool to solve the general problem.
The setup is the following: We are given a (small) primary pool $\mathcal{D}_p$ with 100% accurate labels and large auxiliary pool $\mathcal{D}_a$ with $< 100\%$ accurate labels.
Possible approaches are to use the auxiliary pool for learning a good representation of the primary pool using dictionary learning in a semi-supervised fashion. Alternatively, one may allow the classifier to disregard some samples (out of the auxiliary pool) that lead to a large contribution to the loss. Finally, we can try to correct the labels of the auxiliary pool and use it afterwards as if it were perfectly labeled. The main focus of this chapter will be label correction.
We investigate the solutions in the context of logistic regression due to its analytical simplicity - thus first a reminder.

3.1. Reminder: Logistic regression

Logistic regression works, similarly to a SVM, by spanning a hyperplane in data space and all samples on one side of this hyperplane are predicted to have a certain label while all remaining data points have the opposite label. Thus, it only solves a binary classification problem but obviously this can be extended to multi-class classification via a one-vs-rest scheme. Finding the optimal hyperplane given a binary classification problem can be formulated as maximizing the likelihood $\mathcal{J}$, so

$$\{\boldsymbol{\omega}^*, b^*\} = \arg\max_{\boldsymbol{\omega}, b} \mathbb{E}_{\boldsymbol{x},y} \mathcal{J}(\boldsymbol{\omega}, b, \boldsymbol{x}, y) \tag{3.1}$$

where the likelihood is the probability that the classifier predicts the correct label y of $\boldsymbol{x}$ and, it is given by

$$\mathcal{J}(\boldsymbol{\omega}, b, \boldsymbol{x}, y) \coloneqq \mathcal{P}(y(\boldsymbol{\omega}^T \boldsymbol{x} + b)) \tag{3.2}$$

where $\mathcal{P}$ assigns a probability that sample $\boldsymbol{x}$ has the correct label to the normal distance from the hyperplane. Note that $y \in \{1, -1\}$. The usual choice is $\mathcal{P}(x) \coloneqq \sigma(x) = \frac{1}{1+\exp\{-x\}}$. Therefore, the probability goes to one if the sample is at a large distance from the

hyperplane and on the correct side, it goes to zero when the sample is at a large distance from the hyperplane and on the wrong side and it is 1/2 if it is exactly at the hyperplane. It also has some nice analytical properties.
Since the underlying distribution is in general not known the averaging over the true distribution is replaced by the product of individual likelihoods of a finite number of samples, thus

$$\mathbb{E}_{\boldsymbol{x},y}\mathcal{J}(\boldsymbol{\omega}, b, \boldsymbol{x}, y) \rightarrow \prod_{(\boldsymbol{x}_l, y_l)\in\mathcal{L}} \sigma(y_l(\boldsymbol{\omega}^T\boldsymbol{x}_l + b)). \tag{3.3}$$

Finally, we take the natural logarithm of this expression and end up with the following

$$\{\boldsymbol{\omega}^*, b^*\} = \arg\max_{\boldsymbol{\omega},b} \sum_{(\boldsymbol{x}_l, y_l)\in\mathcal{L}} \ln\sigma(y_l\boldsymbol{\omega}^T\boldsymbol{x}_l + y_l b). \tag{3.4}$$

This is valid since the logarithm is a monotonously increasing function, i.e., $\arg\max f \Leftrightarrow \arg\max \ln f$. In the context of optimization this problem is often rewritten as minimizing a loss function, so

$$\{\boldsymbol{\omega}^*, b^*\} = \arg\min_{\boldsymbol{\omega},b} \sum_{(\boldsymbol{x}_l, y_l)\in\mathcal{L}} \mathcal{L}(\boldsymbol{\omega}, b, \boldsymbol{x}_l, y_l) \tag{3.5}$$

$$= \arg\min_{\boldsymbol{\omega},b} \sum_{(\boldsymbol{x}_l, y_l)\in\mathcal{L}} \ln\left(1 + \exp\left\{-y_l(\boldsymbol{\omega}^T\boldsymbol{x}_l + b)\right\}\right) \tag{3.6}$$

and with our particular choice of σ, the function $\mathcal{L}$ is known as logistic loss.
We can solve this using gradient descent or Newton's method in an iterative manner. After initialization and using Newton's method the update rule is given by

$$\tilde{\boldsymbol{\omega}} \equiv \begin{pmatrix} \boldsymbol{\omega} \\ b \end{pmatrix} \leftarrow \begin{pmatrix} \boldsymbol{\omega} \\ b \end{pmatrix} - \alpha\mathbf{H}^{-1}\nabla_{\boldsymbol{\omega},b}\left(\sum_{(\boldsymbol{x}_l, y_l)\in\mathcal{L}} \ln\mathcal{J}(\boldsymbol{\omega}, b, \boldsymbol{x}_l, y_l)\right) \tag{3.7}$$

where α is a learning rate and $\mathbf{H}$ is the Hessian matrix which is given by

$$H_{d,d'} = \mathbb{E}_{\boldsymbol{x},y}\frac{\partial \ln\mathcal{J}(\tilde{\boldsymbol{\omega}}, \boldsymbol{x}, y)}{\partial\tilde{\omega}_d\partial\tilde{\omega}_{d'}} \rightarrow \sum_{(\boldsymbol{x}_l, y_l)\in\mathcal{L}} \frac{\partial \ln\mathcal{J}(\tilde{\boldsymbol{\omega}}, \boldsymbol{x}_l, y_l)}{\partial\tilde{\omega}_d\partial\tilde{\omega}_{d'}}. \tag{3.8}$$

Given our logistic loss function we can give the expressions for the gradient w.r.t the classifier parameters and the Hessian matrix explicitly. It is helpful to first note that

$$\partial_t\sigma(f(t)) = \sigma(f(t))\sigma(-f(t))\partial_t f(t) \Rightarrow \partial_t \ln\sigma(f(t)) = \sigma(-f(t))\partial_t f(t) \tag{3.9}$$

where f is an at least once differentiable function and also note that

$$\sigma(x) + \sigma(-x) = 1 \quad \forall x \in \mathbb{R}. \tag{3.10}$$

Then, we can easily write down the gradient and Hessian matrix of the approximated log-likelihood function

$$\partial_{\omega_d}\left(\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\ln\mathcal{J}(\boldsymbol{\omega},b,\boldsymbol{x}_l,y_l)\right)=\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\sigma(-y_l(\boldsymbol{\omega}^T\boldsymbol{x}_l+b))y_l x_{l,d}, \tag{3.11}$$

$$\partial_{b}\left(\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\ln\mathcal{J}(\boldsymbol{\omega},b,\boldsymbol{x}_l,y_l)\right)=\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\sigma(-y_l(\boldsymbol{\omega}^T\boldsymbol{x}_l+b))y_l, \tag{3.12}$$

$$\partial_{\omega_d}\partial_{\omega_{d'}}\left(\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\ln\mathcal{J}(\boldsymbol{\omega},b,\boldsymbol{x}_l,y_l)\right)=\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}-\sigma(y_l(\boldsymbol{\omega}^T\boldsymbol{x}_l+b))\sigma(-y_l(\boldsymbol{\omega}^T\boldsymbol{x}_l+b))x_{l,d}x_{l,d'}, \tag{3.13}$$

$$\partial_{\omega_d}\partial_{b}\left(\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\ln\mathcal{J}(\boldsymbol{\omega},b,\boldsymbol{x}_l,y_l)\right)=\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}-\sigma(y_l(\boldsymbol{\omega}^T\boldsymbol{x}_l+b))\sigma(-y_l(\boldsymbol{\omega}^T\boldsymbol{x}_l+b))x_{l,d}, \tag{3.14}$$

$$\partial_{b}\partial_{b}\left(\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\ln\mathcal{J}(\boldsymbol{\omega},b,\boldsymbol{x}_l,y_l)\right)=\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}-\sigma(y_l(\boldsymbol{\omega}^T\boldsymbol{x}_l+b))\sigma(-y_l(\boldsymbol{\omega}^T\boldsymbol{x}_l+b)). \tag{3.15}$$

If this would be the end of the story, then it would be impossible for logistic regression to solve a radial problem such as in figure 4.3. We are still missing the most important trick that is, replacing the projection $\boldsymbol{\omega}^T\boldsymbol{x}$ by an arbitrary kernel function. The projection $\boldsymbol{\omega}^T\boldsymbol{x}$ we have used up to this point is called a linear kernel function. Figure 2.6 displays other candidates.
One of the most important is the radial basis function kernel (rbf) defined as

$$\mathcal{K}_{\text{rbf}}(\boldsymbol{x}_i,\boldsymbol{x}_j)\equiv\exp\left\{-\gamma\|\boldsymbol{x}_i-\boldsymbol{x}_j\|_2^2\right\}\ \text{where}\ \|.\|_2\ \text{denotes Euclidean norm and}\ \gamma>0. \tag{3.16}$$

Replacing $\boldsymbol{\omega}^T\boldsymbol{x}\rightarrow\mathcal{K}_{\text{rbf}}(\boldsymbol{\omega},\boldsymbol{x})$ yields the optimization problem

$$\{\boldsymbol{\omega}^*,b^*\}=\arg\max_{\boldsymbol{\omega},b}\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\ln\sigma(y_l(\mathcal{K}_{\text{rbf}}(\boldsymbol{\omega},\boldsymbol{x}_l)+b)). \tag{3.17}$$

And for completeness we give the gradient and the elements of the Hessian matrix required for optimization using Newton's method explicitly

$$\partial_{\omega_d}\left(\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\ln\sigma(\xi)\right)=\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\sigma(-\xi)(-2\gamma)y_l\mathcal{K}_{\text{rbf}}(\boldsymbol{\omega},\boldsymbol{x_l})(\omega_d-x_{l,d}), \tag{3.18}$$

$$\partial_{b}\left(\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\ln\sigma(\xi)\right)=\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\sigma(-\xi)y_l, \tag{3.19}$$

$$\partial_{\omega_d}\partial_{\omega_{d'}}\left(\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\ln\sigma(\xi)\right)=\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\sigma(-\xi)(-2\gamma)^2y_l\mathcal{K}_{\text{rbf}}(\boldsymbol{\omega},\boldsymbol{x_l})(\omega_d-x_{l,d})(\omega_{d'}-x_{l,d'}) \tag{3.20}$$

$$(1-y_l\mathcal{K}_{\text{rbf}}(\boldsymbol{\omega},\boldsymbol{x_l})\sigma(\xi))\,, \tag{3.21}$$

$$\partial_{\omega_d}\partial_{b}\left(\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\ln\sigma(\xi)\right)=\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\sigma(-\xi)\sigma(\xi)2\gamma\mathcal{K}_{\text{rbf}}(\boldsymbol{\omega},\boldsymbol{x_l})(\omega_d-x_{l,d}), \tag{3.22}$$

$$\partial_{b}\partial_{b}\left(\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}\ln\sigma(\xi)\right)=\sum_{(\boldsymbol{x}_l,y_l)\in\mathcal{L}}-\sigma(-\xi)\sigma(\xi). \tag{3.23}$$

where $\xi \coloneqq y_l(\mathcal{K}_{\text{rbf}}(\boldsymbol{\omega},\boldsymbol{x}_l)+b)$.
This concludes our review of logistic regression and we return to the initial topic - the utilization of the auxiliary pool.

3.2. Dictionary learning

We use the auxiliary pool to learn a sparse representation of our data while optimizing the classifier's loss based on this representation.
We define sparsity by example. Consider the following two Euclidean-unit-vectors

$$\boldsymbol{e}_1=(1,0)^T,\quad \boldsymbol{e}_+=\frac{1}{\sqrt{2}}(1,1)^T. \tag{3.24}$$

We say $\boldsymbol{e}_1$ is sparse and $\boldsymbol{e}_+$ is not sparse. We find (relatively) sparse vectors by regularizing the length of vectors in an optimisation problem through a p-norm with $p<2$. To see this, consider the following optimisation problem of finding Euclidean-unit-vectors

$$\boldsymbol{e}^*=\arg\min_{\boldsymbol{e}}\ (|1-||\boldsymbol{e}||_2|+\lambda||\boldsymbol{e}||_p) \tag{3.25}$$

with $\lambda>0$. It is solved by $\boldsymbol{e}_1$ better than by $\boldsymbol{e}_+$ for $p<2$ and reversed for $p>2$. The following table should make this clear:

$$\begin{array}{lllll} p=1: & ||\boldsymbol{e}_1||_1=1 & ||\boldsymbol{e}_+||_1=\sqrt{2} & \text{induces sparsity} \\ p=2: & ||\boldsymbol{e}_1||_2=1 & ||\boldsymbol{e}_+||_2=1 & — \\ p=\infty: & ||\boldsymbol{e}_1||_\infty=1 & ||\boldsymbol{e}_+||_\infty=\frac{1}{\sqrt{2}} & \text{suppresses sparsity} \end{array} \tag{3.26}$$

The visualisation of different p-norm-unit-circles in figure 2.5 may be helpful.
A sparse representation is desirable since sparse data is assumed to be better separable in input space. This makes the data easier to classify.
The process of finding a sparse representation that is still able to (more or less) reconstruct the original data is called dictionary learning. Its loss is given by

$$\underset{\mathbf{D}\in\mathcal{D},\,\tilde{\boldsymbol{x}}\in\mathbb{R}^K}{\arg\min} \underbrace{\left(\mathbb{E}_{\boldsymbol{x}}\left[\frac{1}{2}\underbrace{||\boldsymbol{x}-\mathbf{D}\tilde{\boldsymbol{x}}||_2^2}_{\text{reconstruction error}}\right]+\lambda||\tilde{\boldsymbol{x}}||_1\right)}_{=\mathcal{L}}. \tag{3.27}$$

where

$$\mathcal{D}=\left\{\mathbf{D}\in\mathbb{R}^{D\,\mathrm{x}\,K}\ \text{s.t.}\ ||\boldsymbol{D}_k||_2\leq 1 \quad \forall k=1\rightarrow K\right\}\ \text{and}\ \lambda\geq 0. \tag{3.28}$$

Now we know what we mean by finding a sparse representation of our data. We then use the data in this representation to optimize a classifier's loss.
The classifier's loss is given by logistic loss defined in the previous chapter (see eq. (3.6)). The optimisation problem in such a semi-supervised dictionary learning is a convex combination of the classifier's loss and the loss of dictionary learning. It is given by

$$\underset{\tilde{\boldsymbol{\omega}}\in\mathbb{R}^{K+1},\,\mathbf{D}\in\mathcal{D},\,\tilde{\boldsymbol{x}}\in\mathbb{R}^K}{\arg\min}\left((1-\mu)\mathbb{E}_{\boldsymbol{x},y}\left[\mathcal{L}(\tilde{\boldsymbol{\omega}},y,\tilde{\boldsymbol{x}}(\boldsymbol{x}))\right]+\mu\left(\mathbb{E}_{\boldsymbol{x}}\left[\frac{1}{2}||\boldsymbol{x}-\mathbf{D}\tilde{\boldsymbol{x}}||_2^2\right]+\lambda||\tilde{\boldsymbol{x}}||_1\right)\right) \tag{3.29}$$

where $\mu\in[0,1]$ and $\lambda\geq 0$. This optimisation problem can be solved by, e.g., block coordinate gradient descent, however an easier implementation is to learn the dictionary first and then the classifier, so

$$\underset{\tilde{\boldsymbol{\omega}}\in\mathbb{R}^{K+1}}{\arg\min}\ \mathbb{E}_{\boldsymbol{x},y}\left[\mathcal{L}(\tilde{\boldsymbol{\omega}},y,\tilde{\boldsymbol{x}}^*(\boldsymbol{x}))\right] \tag{3.30}$$

where

$$\tilde{\boldsymbol{x}}^*=\underset{\mathbf{D}\in\mathcal{D},\,\tilde{\boldsymbol{x}}\in\mathbb{R}^K}{\arg\min}\left(\mathbb{E}_{\boldsymbol{x}}\left[\frac{1}{2}||\boldsymbol{x}-\mathbf{D}\tilde{\boldsymbol{x}}||_2^2\right]+\lambda||\tilde{\boldsymbol{x}}||_1\right). \tag{3.31}$$

Approximating the expectation values over sample distribution allows us to utilize the larger auxiliary pool for learning the dictionary since it does not require the auxiliary pool's faulty labels. Therefore, the final optimisation problem becomes

$$\underset{\tilde{\boldsymbol{\omega}}\in\mathbb{R}^{K+1}}{\arg\min}\sum_{(\boldsymbol{x}_{n_p},y_{n_p})\in\mathcal{D}_p}\mathcal{L}(\tilde{\boldsymbol{\omega}},y_{n_p},\tilde{\boldsymbol{x}}_{n_p}=\mathbf{D}^{*T}\boldsymbol{x}_{n_p}) \tag{3.32}$$

where

$$\mathbf{D}^*=\underset{\mathbf{D}\in\mathcal{D},\,\tilde{\mathbf{X}}\in\mathbb{R}^{K\,\mathrm{x}\,N_a}}{\arg\min}\left(\sum_{\boldsymbol{x}_{n_a}\in\mathcal{D}_a}\left[\frac{1}{2}||\boldsymbol{x}_{n_a}-\mathbf{D}\tilde{\boldsymbol{X}}_{\boldsymbol{x}_{n_a}}||_2^2\right]+\lambda||\tilde{\mathbf{X}}||_1\right) \tag{3.33}$$

and where N_a is the number of samples in the auxiliary pool, $\tilde{\boldsymbol{X}}_{\boldsymbol{x}_{n_a}}$ is the new representation of sample $\boldsymbol{x}_{n_a}$ and $||.||_1$ is meant similar as a Frobenius norm but with $p=1$

instead of $p = 2$.
Finally, we note that this approach is conceptionally similar to transfer learning. The field of transfer learning is about extracting and storing knowledge gained from solving one problem and then transferring it to a related problem. Here, the initial problem is learning a sparse representation and the transferred knowledge is the dictionary.

3.3. M-Logit

Since we now know how to utilize the auxiliary pool to learn a (hopefully) good representation, we continue with what is called "M-Logit". It is short for "Migratory-Logit" and the approach is taken from [18].
The idea is to allow the classifier to put less emphasis on some samples from the auxiliary pool based on the classifier's ability to correctly label them.
To give credit to the paper we follow its notation closely, we denote $\mathcal{D}_p$ and $\mathcal{D}_a$ as the primary and auxiliary pool (just as before) with labels y^p, y^a, respectively. We consider a binary classification problem, i.e., $y^p \in \{1, -1\}$ and $y^a \in \{1, -1\}$. We use logistic regression as our classifier and start by splitting the likelihood into two parts, since their label distribution differs

$$\ln \mathcal{J}(\boldsymbol{\omega}, b) = \mathbb{E}_{\boldsymbol{x},y^p}\left[\ln \sigma(y^p(\boldsymbol{\omega}^T\boldsymbol{x} + b))\right] + \mathbb{E}_{\boldsymbol{x},y^a}\left[\ln \sigma(y^a(\boldsymbol{\omega}^T\boldsymbol{x} + b))\right] \tag{3.34}$$

where $\sigma(t) = \frac{1}{1+\exp\{-t\}}$ and estimate the expectation values using our datasets, so

$$\ln \mathcal{J}(\boldsymbol{\omega}, b) \approx \sum_{n_p=1}^{N_p} \ln \sigma(y^p_{n_p}(\boldsymbol{\omega}^T\boldsymbol{x}^p_{n_p} + b)) + \sum_{n_a=1}^{N_a} \ln \sigma(y^a_{n_a}(\boldsymbol{\omega}^T\boldsymbol{x}^a_{n_a} + b)) \tag{3.35}$$

where N_p and N_a are the number of samples in the primary and auxiliary pool.
Next, we introduce the auxiliary variable $\boldsymbol{\mu} = (\mu_1, .., \mu_{N_a})^T$ whose purpose is to lessen the effect of some samples from the auxiliary pool on the training process

$$\ln \mathcal{J}(\boldsymbol{\omega}, b, \boldsymbol{\mu}) \to \sum_{n_p=1}^{N_p} \ln \sigma(y^p_{n_p}(\boldsymbol{\omega}^T\boldsymbol{x}^p_{n_p} + b)) + \sum_{n_a=1}^{N_a} \ln \sigma(y^a_{n_a}(\boldsymbol{\omega}^T\boldsymbol{x}^a_{n_a} + b + \mu_{n_a})). \tag{3.36}$$

Thus, μ_{n_a} reflects the mismatch of $(\boldsymbol{x}^a_{n_a}, y^a_{n_a})$ with $\mathcal{D}_p$. The larger $y^a_{n_a}\mu_{n_a}$ is, the less sensitive is $\Pr(y^a_{n_a}|\boldsymbol{x}^a_{n_a}, \boldsymbol{\omega}, b, \mu_{n_a})$ to a change in the model parameters $\boldsymbol{\omega}, b$.
This is desirable, since if $(\boldsymbol{x}^a_{n_a}, y^a_{n_a})$ is mismatched with $\mathcal{D}_p$, the model parameters $\boldsymbol{\omega}, b$ cannot make both contributions $\sum_{n_p=1}^{N_p} \ln \sigma(y^p_{n_p}(\boldsymbol{\omega}^T\boldsymbol{x}^p_{n_p} + b))$ and $\ln \sigma(y^a_{n_a}(\boldsymbol{\omega}^T\boldsymbol{x}^a_{n_a} + b))$ large simultaneously. Evidently, choosing a large $y^a_{n_a}\mu_{n_a}$ will make the model parameters less sensitive to $(\boldsymbol{x}^a_{n_a}, y^a_{n_a})$ and allow the model to concentrate the fitting on $\mathcal{D}_p$.
Obviously we have to bound $\|\boldsymbol{\mu}\|$ from above since otherwise it would render $\mathcal{D}_a$ mute.

This leads to the following optimization problem

$$\begin{aligned} \max_{\boldsymbol{\omega},b,\boldsymbol{\mu}} \quad & \ln \mathcal{J}(\boldsymbol{\omega},b,\boldsymbol{\mu}) \\ \text{where} \quad & \sum_{n_a=1}^{N_a} y_{n_a}^a \mu_{n_a} \leq CN_a,\ C \geq 0 \\ \text{and} \quad & y_{n_a}^a \mu_{n_a} \geq 0 \quad \forall n_a = 1 \rightarrow N_a. \end{aligned} \tag{3.37}$$

The classifier that results from the optimization problem eq. (3.37) is called "Migratory-Logit" (or "M-Logit").
The hyperparameter C depends on the mismatch between $\mathcal{D}_p$ and $\mathcal{D}_a$, a good starting value is $C = 12\frac{N_m}{N_a}$ where N_m is the estimated number of incorrectly labeled samples in the auxiliary pool (for more information see [18]).
We can solve this problem in a block coordinate iterative process, first optimizing w.r.t. $\boldsymbol{\omega}, b$ while keeping $\boldsymbol{\mu}$ constant and then optimizing w.r.t. $\boldsymbol{\mu}$ while keeping $\boldsymbol{\omega}, b$ constant. To optimize w.r.t. $\boldsymbol{\omega}, b$ we can deploy standard gradient descent with sufficiently small step size. The solution to the problem (3.37) when the model parameters are kept constant is given in analytical form in [18]. A visual interpretation of the analytical solution is given in figure 3.1.
However, the analytical solution is based on the analytical form of the loss function of logistic regression. This limits "M-Logit" to problems that are solvable by logistic regression. Therefore, we generalize this concept of neglecting certain contributions to the loss function to a Multi-Layer perceptron but it applies to any loss-driven learning process where the loss is based on M-estimation.

3.4. MLP with adapting sample weights

Every classifier that minimizes the expected loss or risk by adjusting the model parameter appropriately has to estimate the loss through the provided training dataset. If the estimated loss is a sum over individual losses, this is called M-estimation. We can lessen the contribution of single samples by weighting the individual losses. The estimated loss can then be written as

$$\hat{\mathcal{L}}(\boldsymbol{\theta},\boldsymbol{w}) = \sum_{l=1}^{L} w_l \mathcal{L}(y_l|\boldsymbol{x}_l,\boldsymbol{\theta}) - \lambda_1||\boldsymbol{w}||_1 + \lambda_2||\boldsymbol{\theta}||_2^2 \tag{3.38}$$

where $w_l, \lambda_1, \lambda_2 \geq 0 \quad \forall l = 1 \rightarrow L$ and L is the number of samples in $\mathcal{L}$ and $\mathcal{L}$ is the classifier's loss function.
The motivation is that, by choosing a well-balanced λ_1 the MLP will learn which samples cannot achieve a low loss, since they are incorrectly labeled, and this constant loss will outweigh the regularization term $\propto \lambda_1$ that increases when decreasing a weight.
The task is to *minimize* the estimated, expected loss w.r.t. $\boldsymbol{\theta}, \boldsymbol{w}$ *simultaneously*, so

$$\boldsymbol{\theta}^*, \boldsymbol{w}^* = \arg\min_{\boldsymbol{\theta},\boldsymbol{w}} \hat{\mathcal{L}}(\boldsymbol{\theta},\boldsymbol{w}). \tag{3.39}$$

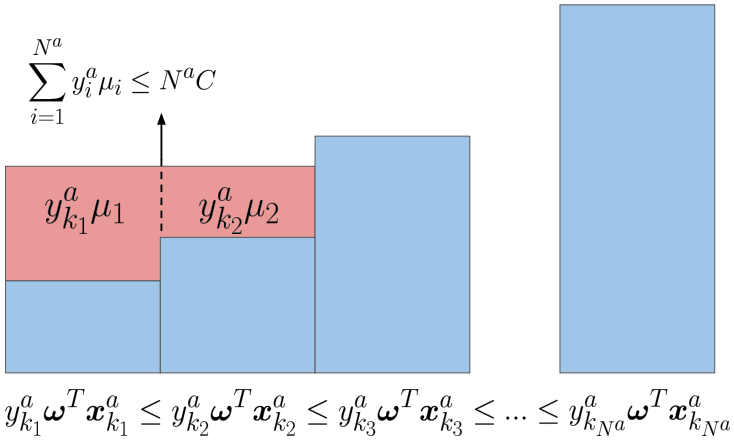


Figure 3.1.: Analytical solution to optimize equation (3.37) w.r.t. $\boldsymbol{\mu}$. For sample $\boldsymbol{x}_{n_a}^a \in \mathcal{D}_a$ the "wealth" is given by $y_{n_a}^a \boldsymbol{\omega}^T \boldsymbol{x}_{n_a}^a$. The blue bars represent the sorted wealth of all samples in $\mathcal{D}_a$ where k_i is a permutation. The strategy is as follows, we assign the least wealthy sample $\boldsymbol{x}_{k_1}^a$ an auxiliary variable μ_1 such that it becomes as wealthy as the second least wealthy, i.e., $y_{k_1}^a \boldsymbol{\omega}^T \boldsymbol{x}_{k_1}^a + y_{k_1}^a \mu_1 = y_{k_2}^a \boldsymbol{\omega}^T \boldsymbol{x}_{k_2}^a$. Then, we increase the wealth of the two least wealthy till they reach the wealth of the third least wealthy. We continue till $\sum_{n_a=1}^{N_a} y_{n_a}^a \mu_{n_a} \leq C N_a$. We set $b = 0$ for notational simplicity.

We solve the optimization in eq. (3.39) iteratively using coordinate descent with sufficiently small step size.

3.5. Label correction

Finally, the last approach to exploiting the auxiliary pool is, to try to correct as many false-labeled samples and treat the auxiliary pool as primary pool afterwards, i.e., treat the auxiliary pool as if it was perfectly labeled.

The following four label correction methods can be divided into two subcategories that correspond to different approaches conceptionally. The first subcategory is based on a self-amplifying classifier. This means training a classifier on the data and using the trained classifier to correct some labels. Then, retrain the classifier on the modified labels and so on. The second subcategory is based on clustering methods to use the structure of the data and correct labels.

3.5.1. Nearest neighbour correction (NNC)

This novel approach is based on a self-amplifying classifier. The idea is to iterate through every sample and classify it using the K-nearest neighbours. If the classifier predicts a different label with a confidence higher than some threshold $p \in (0, 1]$, store the predicted label and mark the sample. After iterating through all samples once, all marked samples adopt their predicted label. Repeat with updated labels till there are no more changing labels.
The procedure is outlined in detail in algorithm 20.

3.5.2. Automatic data enhancement (ADE)

The approach is based on a self-amplifying classifier and is following closely the original paper [36]. The idea is as follows: First, introduce a small uncertainty in the initially given label distribution of each sample, so, e.g., $(\boldsymbol{x}_1, y_1 = 0) \to (\boldsymbol{x}_1, y_1 = \arg\max\, \boldsymbol{p}_1, \boldsymbol{p}_1 = (0.95, 0.05))$ for binary classification. Then, train a shallow (one hidden layer) MLP with only one hidden node for one epoch and small step size. Next, use the trained MLP to predict the label distribution of every sample and update every samples label distribution doing a small step towards the prediction. Update the labels according to $y_i = \arg\max\, \boldsymbol{p}_i$. Note that during the training of the MLP the target is *not* the label distribution. Repeat this process N_{epoch} times, then calculate an extended loss $\mathcal{L}^{(\text{adj})}$. This repetition N_{epoch} times is called an era. Repeat eras till the extended loss (which we calculate after each era) converges and store the last extended loss and the current labels. Increase the number of hidden nodes by one and repeat this process all over again. Stop increasing the number of hidden nodes till the converged extended loss increases twice in succession. Then, return the label configuration corresponding to the

Algorithm 20 Nearest neighbour correction

Input: Auxiliary pool $\mathcal{D}_a$
Parameters: Number of neighbours to consider $K \in \mathbb{N}$, confidence $p \in (0, 1]$
Output: Corrected pool $\mathcal{D}_a$

while $N_c \geq 1$ **do**
  $N_c = 0$
  copy $\mathcal{D}_a$ to $\mathcal{D}_{a,t} = \{(\boldsymbol{x}_{t,1}, y_{t,1}), (\boldsymbol{x}_{t,2}, y_{t,2}), .., (\boldsymbol{x}_{t,N_a}, y_{t,N_a})\}$
  for $n_a = 1 \rightarrow N_a$ **do**
    find K-nearest neighbours $\{(\tilde{\boldsymbol{x}}_1, \tilde{y}_1), .., (\tilde{\boldsymbol{x}}_K, \tilde{y}_K)\} \subseteq \mathcal{D}_a$ of $\boldsymbol{x}_{n_a} \in \mathcal{D}_a$ # note that the labels of the NN are taken from $\mathcal{D}_a$ and not $\mathcal{D}_{a,t}$
    let $\hat{\boldsymbol{p}}$ be a zero-vector of length Z
    for $k = 1 \rightarrow K$ **do**
      $\hat{p}_{\tilde{y}_k}$ += 1
    end for
    $\hat{\boldsymbol{p}} \leftarrow \frac{\hat{\boldsymbol{p}}}{K}$ # normalize
    $t = \max_{z \in \mathcal{Z}} \hat{p}_z$ and $z^* = \arg\max_{z \in \mathcal{Z}} \hat{p}_z$
    if $t \geq p$ and $y_{n_a} \neq z^*$ **then**
      $y_{t,n_a} = z^*$
      N_c += 1
    end if
  end for
  $\mathcal{D}_a \leftarrow \mathcal{D}_{a,t}$ # carry over all changes to next iteration
end while
return $\mathcal{D}_a$

lowest converged, extended loss. This is the corrected label assignment.
The procedure is also outlined in algorithm 21.

A key point is that the extended loss $\mathcal{L}^{(\text{adj})}$ is the sum of three contributions. First, the loss of the MLP, then an additional loss term that increases as the number of hidden nodes increases. This is necessary to punish a model that is able of stronger overfitting (since the model is more complex) which is especially unfortunate if some of the training data is incorrect. Finally, the extended loss has one more term that increases as the class ratio, resulting from the label assignment after each era, more strongly deviates from the initial class ratio. This means if the initial label configuration represents a balanced dataset then the corrected label configuration should be (more or less) balanced as well. This is a consequence of assuming that the labels were mislabeled randomly.
At this point we stop and don't dive into the details but you may refer to the paper [36] for more information or, alternatively, look at my implementation in code, see section 3.5.5.

3.5.3. Cluster correction (CC)

As the name suggests this method is based on clustering the data and it is a slight modification to the one proposed in [22].
The concept is to assign every sample the label distribution of its main cluster, i.e., samples in one cluster have the same label distribution. By introducing homogeneity we hope to eliminate the false labels. To create cluster diversity we vary the number of cluster centroids and every sample's final label distribution is summed over the label distribution that results from the different number of cluster centroids. Lastly, every sample's label is the argmax of its final label distribution.
The entire procedure is outlined in algorithm 22.

3.5.4. Binary cluster correction (BCC)

This is again a correction method based on clustering which is taken from [24]. To understand it we first discuss its underlying assumption.
In a perfect world instances of the same class are clustered around some region in data space and different clusters do not overlap. This implies that if instances are labeled perfectly the samples of one class can be approximated using only one clustering centroid. But for every incorrectly labeled sample this cluster centroid would be a poor approximation since class clusters do not overlap. This is the core idea behind BCC which works in detail as follows. First, the auxiliary pool is split into subsets where each subset is made up from instances that belong to the same class. Then, K-Means is used to fit two cluster centroids to each subset since every subset consists of correct and incorrect labels. For every subset the larger of the two clusters (the centroid to which more samples belong/are closer) is assumed to be the "correctly labeled"-cluster and all its samples are marked. Next, a classifier, e.g., a K-nearest neighbour classifier, is

Algorithm 21 Automatic data enhancement

Input: Auxiliary pool $\mathcal{D}_a$

Parameters: number of epochs per era $N_{\text{epoch}} \in \mathbb{N}$,
maximal number of eras $N_{\text{era}} \in \mathbb{N}$,
learning rate model parameters $\eta_{\boldsymbol{\omega}} \in (0, 1]$,
learning rate label distribution $\eta_{\boldsymbol{p}} \in (0, 1]$,
convergence criterion $\epsilon \in (0, 1]$

Output: Corrected pool $\mathcal{D}_a$

let $h = 1$ be the initial number of hidden nodes of the MLP
let $\boldsymbol{g}^{(\text{init})} = \mathit{Label\text{-}Distribution}(\mathcal{D}_a)$ be the vector of initial class ratios
while True **do**
  copy and extend $\mathcal{D}_a$ to $\mathcal{D}_{a,h} = \{(\boldsymbol{x}_{h,1}, y_{h,1}, \boldsymbol{p}_{h,1}), .., (\boldsymbol{x}_{h,N_a}, y_{h,N_a}, \boldsymbol{p}_{h,N_a})\}$
  for $n_{\text{era}} \to N_{\text{era}}$ **do**
    for $n_{\text{epoch}} \to N_{\text{epoch}}$ **do**
      train MLP for one epoch with step size $\eta_{\boldsymbol{\omega}}$ on

$$\{(\boldsymbol{x}_{h,1}, y_{h,1}), .., (\boldsymbol{x}_{h,N_a}, y_{h,N_a})\} \subset \mathcal{D}_{a,h}$$

      use MLP to predict label distribution

$$\hat{\boldsymbol{p}}_{h,n_a} \coloneqq \mathcal{P}(\mathcal{F} = \text{MLP}, \boldsymbol{x}_{h,n_a}) \quad \forall n_a = 1 \to N_a$$

      update label distributions, $\boldsymbol{p}_{h,n_a} \mathrel{+}= \eta_{\boldsymbol{p}}(\hat{\boldsymbol{p}}_{h,n_a} - \boldsymbol{p}_{h,n_a}) \quad \forall n_a = 1 \to N_a$
      update label assignments, $y_{h,n_a} \leftarrow \arg\max\ \boldsymbol{p}_{h,n_a} \quad \forall n_a = 1 \to N_a$
    end for
    calculate extended loss $\hat{\mathcal{L}}_h^{(\text{adj})}(n_{\text{era}}) = \mathcal{L}^{(\text{adj})}$ where

$$\mathcal{L}^{(\text{adj})} \coloneqq \mathcal{L}^{(\text{MLP})} + \mathcal{L}^{(\text{complexity})}(h) + \mathcal{L}^{(\text{deviation})}(\boldsymbol{g}^{(\text{init})}, \boldsymbol{g}^{(\text{current})})$$

    if $\left(|\hat{\mathcal{L}}_h^{(\text{adj})}(n_{\text{era}}) - \hat{\mathcal{L}}_h^{(\text{adj})}(n_{\text{era}} - 1)| \leq \epsilon\right)$ or (n_{era} is N_{era}) **then**
      store the last loss $\hat{\mathcal{L}}_h^{(\text{adj})} \leftarrow \hat{\mathcal{L}}_h^{(\text{adj})}(n_{\text{era}})$ and current label configuration $\mathcal{D}_{a,h}$
    end if
  end for
  if $\left(\hat{\mathcal{L}}_h^{(\text{adj})} \geq \hat{\mathcal{L}}_{h-1}^{(\text{adj})}\right)$ and $\left(\hat{\mathcal{L}}_{h-1}^{(\text{adj})} \geq \hat{\mathcal{L}}_{h-2}^{(\text{adj})}\right)$ **then**
    break # exit while-loop
  end if
  $h \mathrel{+}= 1$
end while
$h^* = \arg\min_h\ \hat{\mathcal{L}}_h^{(\text{adj})}$
return $\mathcal{D}_{a,h^*}$

Algorithm 22 Cluster correction

Input: Auxiliary pool $\mathcal{D}_a$
Parameters: number of clusterings $A \in \mathbb{N}$
Output: corrected pool $\mathcal{D}_a$

let $\mathbf{W}$ be a zero-array of shape (N_a, Z) # number of classes Z
let $\boldsymbol{g}^{(\text{init})} = \mathit{Label\text{-}Distribution}(\mathcal{D}_a)$ be the vector of initial class ratios
for $a = 1 \to A$ **do**
  $K \coloneqq \frac{a}{A}\frac{N}{2} + 2$
  use K-Means to fit K cluster centroids
  compute $\boldsymbol{m}$ with $m_{\boldsymbol{x}_{n_a}} \in \{1, .., K\}$ the main cluster of sample $\boldsymbol{x}_{n_a}$ $\quad \forall \boldsymbol{x}_{n_a} \in \mathcal{D}_a$
  let $\boldsymbol{g}_k$ be the vector of class ratios of the samples with main cluster k $\quad \forall k = 1 \to K$
  let N_k be the number of samples with main cluster k $\quad \forall k = 1 \to K$
  for $n_a = 1 \to N_a$ **do**
    let $k \coloneqq m_{\boldsymbol{x}_{n_a}}$
    calculate $f = \min\left(2, \log_{10} N_k\right)$
    $\boldsymbol{W}_{\boldsymbol{x}_{n_a}}$ += $f \frac{\boldsymbol{g}_k - \boldsymbol{g}^{(\text{init})}}{\boldsymbol{g}^{(\text{init})}}$ # element-wise
  end for
end for
$y_{n_a} = \arg\max_{z \in \mathcal{Z}} W_{\boldsymbol{x}_{n_a}, z}$ $\quad \forall n_a = 1 \to N_a$
return $\mathcal{D}_a$

trained on all marked samples. Finally, treat all unmarked samples as unlabeled and use the trained classifier to predict those labels. The predictions are the corrected labels.

3.5.5. Implementation in code

To see an implementation in python of all algorithms of section 3.5 you may check out my repository called "automatic_label_correction" on Github

`https://github.com/SimiPixel/automatic_label_correction`.

4. Results on toy data

After all this theory it is time to put it to use, so in this section we apply active learning and label correction to several toy and benchmark datasets. First, we create our own 2D-dataset with different geometric decision boundaries to gain intuition about the query strategies of pool-based active learning. Then, we proceed with typical benchmark datasets such as Iris, Wine, EMNIST-Digits and the more demanding MNIST-Fashion and investigate the performance gain by deploying active learning. In particular, we take a look at the following query strategies: RS, US, EER, NNC, CMS, MdS, ALBL and DEAL. To access the effectiveness of label correction we artificially falsify the toy datasets and try to reverse this process. Here, we investigate all four label correction methods, namely: NNC, ADE, CC and BCC.
Regardless, we start by introducing the toy datasets in more detail.

4.1. Toy datasets

4.1.1. 2D-input with geometric decision boundaries

For a first test we set up a toy classification problem. We draw 200 samples randomly from a constant pdf with non-zero density only inside a 2-dimensional unit-square. Further, we draw 200 more samples from a two-dimensional Gaussian pdf with expectation value and covariance matrix

$$\boldsymbol{\mu} = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} 0.05 & 0 \\ 0 & 0.05 \end{pmatrix}. \tag{4.1}$$

To be more precise, in the case of the Gaussian pdf, we draw as long as we have less than 200 points within a unit square, and then disregard those outside of the unit square. The points are displayed in figure 4.2.
We construct a binary classification problem by assigning labels w.r.t. a circle and a XOR decision boundary. The choice of these exact datasets is motivated by [29], and similar toy datasets there. So we end up with four different sets of data, see figure 4.3.

4.1.2. Iris

The Iris dataset contains three classes of different subgroups of the flower species “Iris” with 50 samples for each class. Every instance is characterized by four positive real

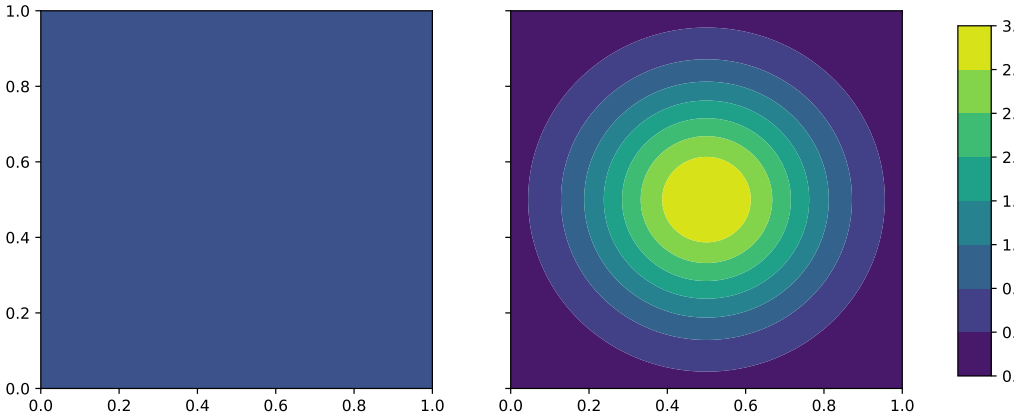


Figure 4.1.: The samples for the 2D-input are drawn from two distributions - a Gaussian- and a constant/uniform pdf.

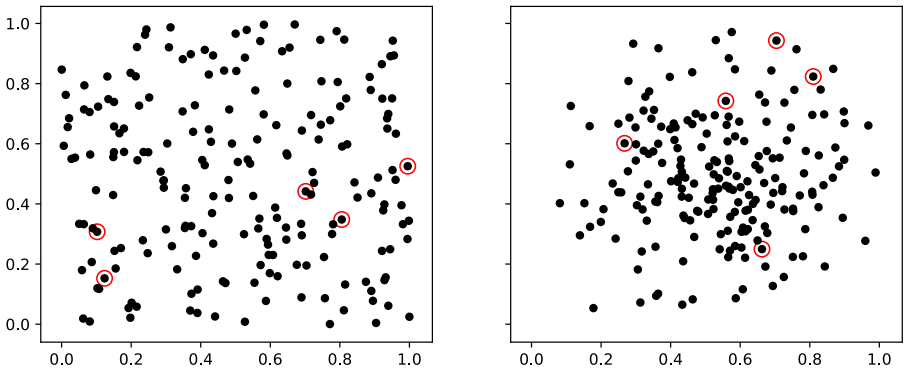


Figure 4.2.: 200 points drawn from a constant- and Gaussian probability density function. The marked points is the initial data for the active learner in figure 4.6.

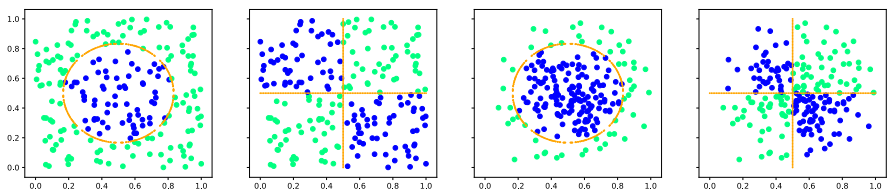


Figure 4.3.: Four different datasets - drawn from either a constant- or Gaussian pdf and labeled with either a circular or XOR decision boundary.

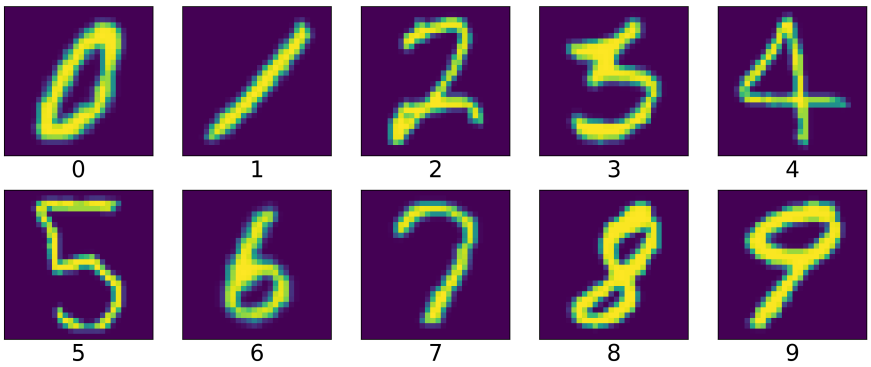


Figure 4.4.: The ten classes of EMNIST-Digits.

values corresponding to the sepal length and -width and to the petal length and -width of the flower - values are in cm.

4.1.3. Wine

The Wine dataset involves 178 samples divided into three classes (class balance: $\{59, 71, 48\}$) with a dimensionality of 13 real positive values corresponding to various wine related characteristic values such as, e.g., alcohol concentration, phenols or flavonoids. This data is the result of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars [12]. The task is to predict the cultivar given the chemical analysis.

4.1.4. EMNIST-Digits and MNIST-Fashion

The EMNIST-Digits dataset [6] and the MNIST-Fashion dataset from zalando research [33] are similar datasets. EMNIST-Digits is a dataset containing 280.000 pictures of handwritten digits with ten balanced classes whereas MNIST-Fashion involves 70.000 pictures of fashion products with ten balanced classes. In both cases every picture consists of 28x28 pixels and each pixel has a color value between zero and 255. The ten classes are displayed in figure 4.4 and figure 4.5.
From both datasets we also generate a binary classification problem by reducing it to two classes - threes or fives and dresses or coats, respectively. These combinations seem to be the most difficult to differentiate between.

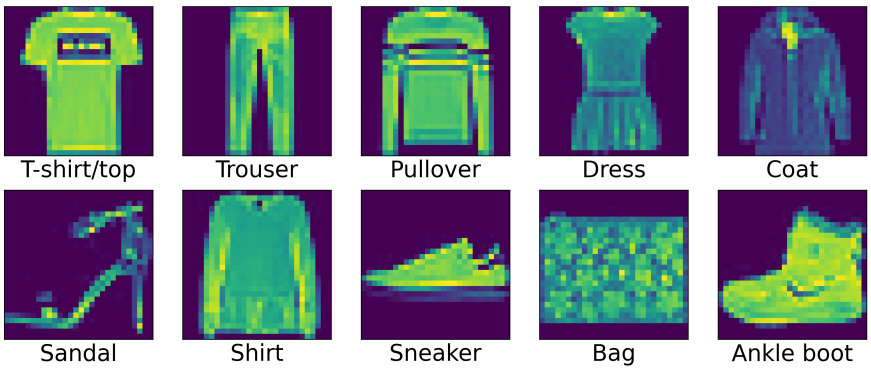


Figure 4.5.: The ten classes of MNIST-Fashion.

Characteristic features of toy datasets:

Dataset name	2D-input	Iris	Wine	EMNIST-Digits	MNIST-Fashion
Number of samples	200	150	178	280000	70000
Number of features	2	4	13	784	784
Number of classes	2	3	3	10	10

4.2. Pool-based active learning

In this section we apply pool-based active learning, see section 2.1, to the datasets mentioned in the previous section 4.1. To gain intuition why active learning works we start with some nicely visualisable 2D-input, then we continue with the other toy datasets and two different classifiers: A support vector machine (SVM) and a random forest classifier (RFC).

4.2.1. 2D-input with geometric decision boundaries

We use pool-based active learning on the four datasets displayed in figure 4.3. For all four datasets we choose a SVM with radial basis function kernel, see eq. (3.16), and $\gamma = 15$ as our classifier.
It is noteworthy that the choice of γ strongly impacts the classifier's performance. The classifier performs poorly for low values of γ. This makes sense since the average distance between two points in the four datasets is small, a small γ makes the already small distance even smaller.
We compare four different base query strategies (base query strategies = strategies from section 2.1.4), namely RS, US, EER and NNC. Further, we compare two more query

strategies that both combine RS and US, namely, ALBL and DEAL.
In the case of EER we have to fix the classifier's random seed to make the certainty gains (or losses), after adding different points, comparable.
To compare the performance gain of different query strategies we split each dataset into 100 training and 100 test samples. Every active learner (query strategy + classifier) starts with five randomly chosen labeled samples from the training pool as initial data. We choose five points because of the implementation of EER - it simply considers every occurring label in the labeled pool a possible label. This means if the labeled pool consists of only one label EER will not work as expected. Obviously we could have fixed the possible labels beforehand at the cost of making EER less easy to use in general.

Now, the boundary conditions are clear and we can continue by discussing several results. The reason for creating the four datasets and applying pool-based active learning to them is to gain intuition behind the workings of the different query strategies. For this task such a 2D-input is well suited since it is nicely visualisable. In this spirit we consider the "snapshot" in figure 4.6. It displays the "state" of the active learner that always queries the sample with the highest entropy (US) after 50 training samples. To be more precise, after the query strategy US has proposed a sample to label and the sample has been added to the labeled pool for 45 iterations. Remember that all query strategies start with the same five initial samples.
The first row in figure 4.6 is the classifier's (and active learner's) labeled pool where the marked points are initial points, these are also the marked points in figure 4.2. The second row is the classifier's current prediction in input space and the third row is the utility score of the query strategy, here entropy, in input space. Dark regions correspond to high entropy.
We can nicely see that the classifier is the most uncertain at its current decision boundary. This was expected but it is still nice that intuition is confirmed.
The four columns are the four different datasets. In the first and third column we also see that the active learner avoided to query points from the center region (since the decision boundary is not going through the center region) and thereby induces sampling bias, see section 2.1.3. We say that there is sampling bias because, e.g., in the third column and first row we see that the labeled pool has no samples in the center region even though the sample density is largest in the center region in the entire labeled dataset, see third column of figure 4.3.

Next, we discuss if and how well the six different query strategies are able to improve label complexity. Remember that the label complexity refers to the number of labeled data/labels necessary to reach a certain test accuracy.
One run consists of all six query strategies successively querying samples to label and adding them to the training pool for 95 iterations. At every iteration we use the currently labeled data to train the same classifier and compute the test accuracy on the same, 100 samples large, test pool. All query strategies start with the same five labeled samples. We randomly redraw the four datasets for every run as we want to compare the performance specific to the classification problem but not the actual data points.

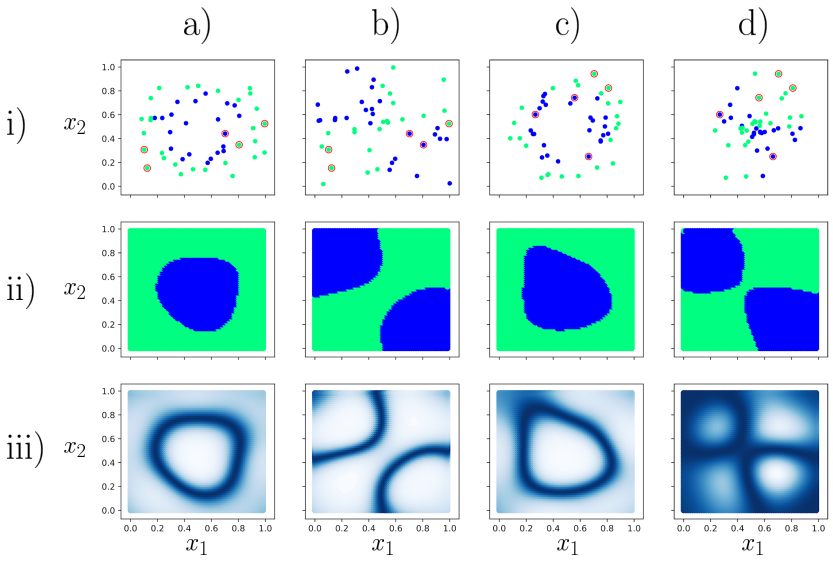


Figure 4.6.: State of active learner that queries samples with highest entropy (US) for 45 iterations. We start with five initial points (marked as red), then the labeled pool consists of 50 training samples in total. The four columns a)-d) correspond to the four datasets displayed in figure 4.3. The first row i) is the labeled pool of the classifier/active learner, the second row ii) is the classifier's label prediction in input space - the color indicates the predicted label in a certain region in input space. The third row iii) is the entropy of each point in input space - white color corresponds to low entropy. We can see that the samples closest to the classifier's decision boundary have the highest entropy.

We average the test accuracy at each query iteration and for every query strategy over 50 runs. This gives us the plots in figure 4.7.
Let's discuss the results for the different query strategies one by one.
RS corresponds to not having a query strategy and sets the baseline for label complexity, i.e., what accuracy we can "easily" achieve given a certain number of labels.
In all four datasets US always performs worse than RS early on (with few labeled samples) and becomes the best query strategy later on (with numerous labeled samples). As already mentioned, US chooses the sample with the highest uncertainty (here entropy) to be queried next. This is the sample that is close to the decision boundary but the decision boundary implied by the *classifier*. Therefore, early on, the estimated decision boundary is still vague. This leads to the query of below average samples. Later on, as the classifiers estimated decision boundary coincides with the true decision boundary US will become a good query strategy. This is typical of an exploit-heavy query strategy, a query strategy that is prone to induce sampling bias. Either way US is capable of heavily improving label complexity. E.g., in case of the "Gaussian + circle" dataset, using US the accuracy gain converges at ≈ 40 labels, whereas randomly sampling requires all 100 samples to be labeled to reach the same accuracy level.
Unsurprisingly, EER behaves like an exploit-heavy query strategy, so poorly early and decent later on. That is, because EER estimates the *expected* uncertainty reduction when a sample is added to the labeled pool. The more labeled data the better the estimation of the expectation value, i.e., the better the prediction of the class membership probabilities.
NNC, in contrast to US and EER, performs exceptionally well early on and poorly later (but still better than RS). This opposing behaviour motivates referring to NNC as an explorative query strategy. This coincides with the fact that NNC does not use/exploit any knowledge (the labels) acquired during the learning. NNC turns out to be a good, systematic way of exploring data space and it synergises well with an exploit heavy query strategy such as US that is used when sufficient labeled samples are obtained and exploration fades into exploitation. We can determine this transition point dynamically using a bandit algorithm such as ALBL or DEAL.
On this note we take a look at the performance of ALBL and DEAL. Both combine RS and US. Since US performs worse than RS early on we expect both bandit algorithms to adapt to this and perform as good as RS early on. As soon as US outperforms RS we expect a weight shift that puts emphasis on US thereby leading to ALBL and DEAL performing as good as US later on. This gives us the best of both worlds, so to speak. Taking a closer look at figure 4.7 we find ALBL and DEAL exceeding expectation and significantly surpassing the accuracy of US at the transition point (at roughly 20 labels) in three of the four datasets (not in case of the lower left plot). Comparing ALBL to DEAL we find both to perform comparably with a slight advantage for ALBL.
Finally, we note that in figure 4.7, and as always, the query strategies merely rearrange a limited training pool (of in this case 100 samples). This explains the fact that given a dataset all query strategies finish/end up with the same test accuracy. This happens, because at the last query iteration (so at the far right in each subplot in figure 4.7) all

query strategies have labeled every sample and finish with the same training pool.

As closing words we again emphasise the meaning of the results of this section. We have shown that pool-based active learning is able to greatly decrease label complexity in case of simple 2D-input. E.g. in case of US by $\approx 60\%$. In section 2.1.4 we tried to illuminate the workings behind every query strategy. This typically involved assumptions. E.g. in case of US that samples with high uncertainty in its class membership probability prediction are in fact close to a decision boundary. Or, in case of NNC that a sample's neighbour is actually representative of the sample. By showing that US and NNC perform well we conclude that their respective assumptions are justified for *these datasets*. The next step is to see if the assumptions of, e.g., US and NNC still hold up for more complex datasets.

Parameter sheet for figure 4.7

Datasets: constant + circle (1), constant + XOR (2), Gaussian + circle (3), Gaussian + XOR (4)

Dataset Number	1	2	3	4
Number of samples	200	200	200	200
Number of features	2	2	2	2
Number of classes	2	2	2	2
Initial number of samples	2	2	2	2
Initial number of classes	2	2	2	2
Final number of training samples	100	100	100	100
Number of test samples	100	100	100	100
Number of runs for averaging	100	100	100	100
Number of runs for EER	25	25	25	25

Classifier: SVM with radial basis function kernel and $\gamma = 15$

Query strategies:

	Parameters
RS	-
US	method = 'entropy', model = main classifier
EER	depth = 1, model = main classifier
NNC	-
ALBL	query_strategy = {US}, uniform_sampler = True, model = main classifier, T = 98, $\delta = 0.1$
DEAL	query_strategy = {RS, US}, model = main classifier, $\gamma = 0.1$, $\Delta_T = 10$, T = 98, $\alpha = 0.1$, $\beta = 10$

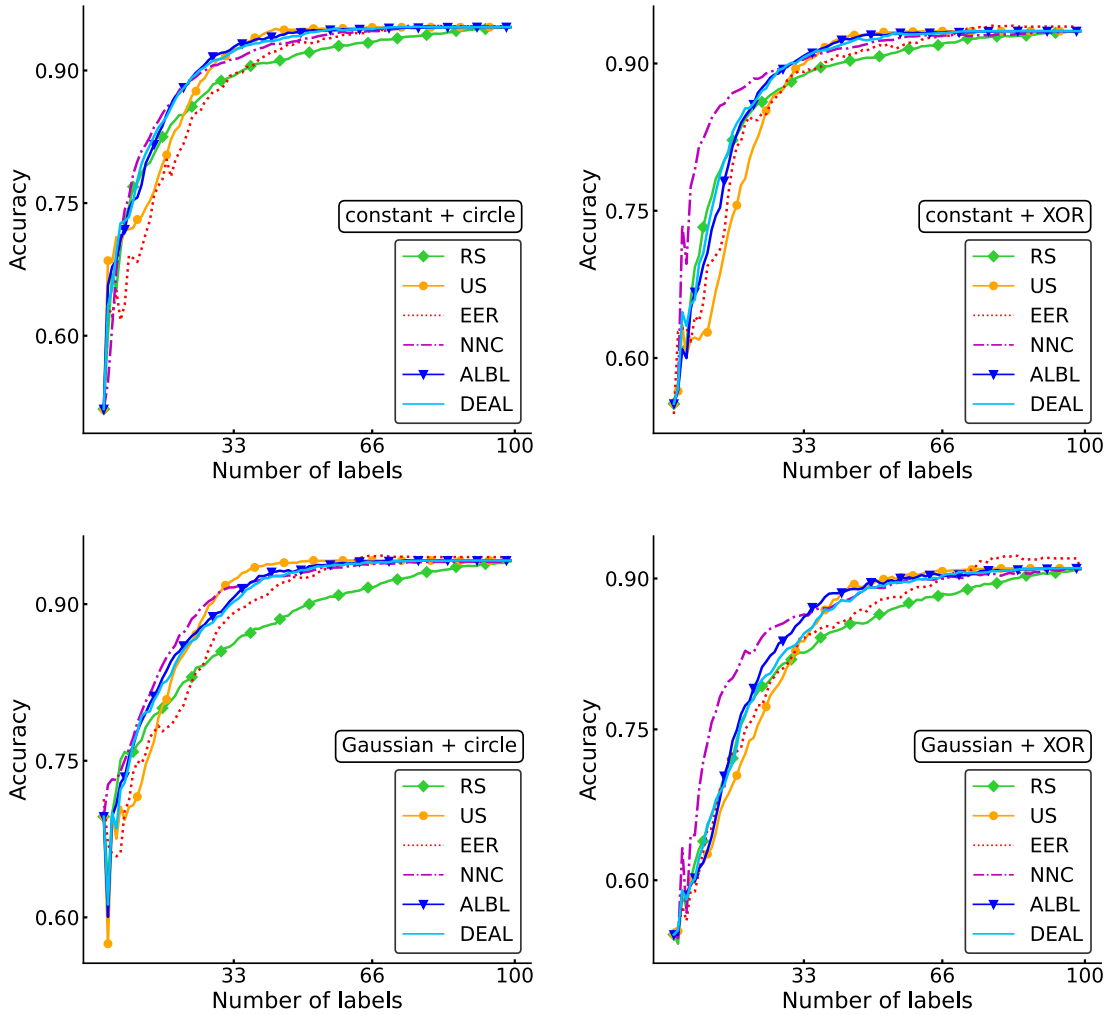


Figure 4.7.: Test accuracy of a SVM with rbf kernel as a function of the number of labels. We compare six different query strategies for the four datasets displayed in figure 4.3. Both ALBL and DEAL combine RS and US. Averaged over 100 runs.

4.2.2. Support vector machine (SVM)

We hope to reproduce the promising results from the previous section on the more complex datasets Iris, Wine, EMNIST-Digits and MNIST-Fashion using a linear SVM as classifier.
We compare five base query strategies, namely RS, US, CMS, EER and MdS. We also include ALBL and DEAL which both combine RS and US.
Since, we want to compare the decrease in label complexity using these query strategies, we need to compute average test accuracies. Therefore, we define one run as the following: For the Iris dataset we randomly choose 100 samples as maximum training pool. The residual 50 samples form the test pool. From the chosen 100 samples that form the maximum training pool, we again randomly choose three samples as initial data. Analogous, for the Wine dataset we randomly choose 100/78/3 samples as maximum training pool/test pool/initial data. For the EMNIST-Digits and MNIST-Fashion dataset we randomly choose 150/300/25 samples as maximum training pool/test pool/initial data. We ensure that all initial datasets contain every class at least once.
Then, all query strategies successively query samples to label and add them to the training pool till the maximum training pool is reached. At every iteration, for every query strategy and for every dataset, we use the currently labeled data to train the classifier and compute the test accuracy.
We average the test accuracy over 20-200 runs (see parameter sheet for figure 4.8). This gives us figure 4.8 with all parameters given in the corresponding parameter sheet.

Let's discuss the results for the different datasets one by one.
For the Iris dataset we find US, CMS, ALBL and DEAL greatly decreasing label complexity. E.g., CMS and US both peak at ≈ 40 labels. In contrast, MdS and EER perform far worse than RS and almost double the label complexity. The reason why CMS and US work well is due to the strong cluster structure of the Iris dataset. The fact that EER performs so poorly is surprising and leaves room for speculation. We note that, early on, US actually increases the label complexity and only outperforms RS after a certain number of labels. We already discussed this familiar behaviour in the previous section.
In case of the Wine dataset US, ALBL and DEAL perform very well. US peaks at ≈ 35 labels. This means that 35 samples chosen by US are sufficient and discarding the remaining 65 samples does not lead to any information loss. MdS performs slightly better than RS. CMS only manages to decrease label complexity later on while EER always increases label complexity.
We discuss the results on the EMNIST-Digits and MNIST-Fashion together. In both cases US, ALBL and DEAL decrease label complexity. E.g. for the EMNIST-Digits dataset US decreases label complexity by $\approx 20\%$. CMS and EER perform comparably to RS. MdS increases label complexity for both datasets.
In case of their binary counterparts we find ALBL and DEAL performing best. Both decrease label complexity. US is a second best due to it suffering from underwhelming performance early on. It is quite remarkable how much better ALBL performs than US,

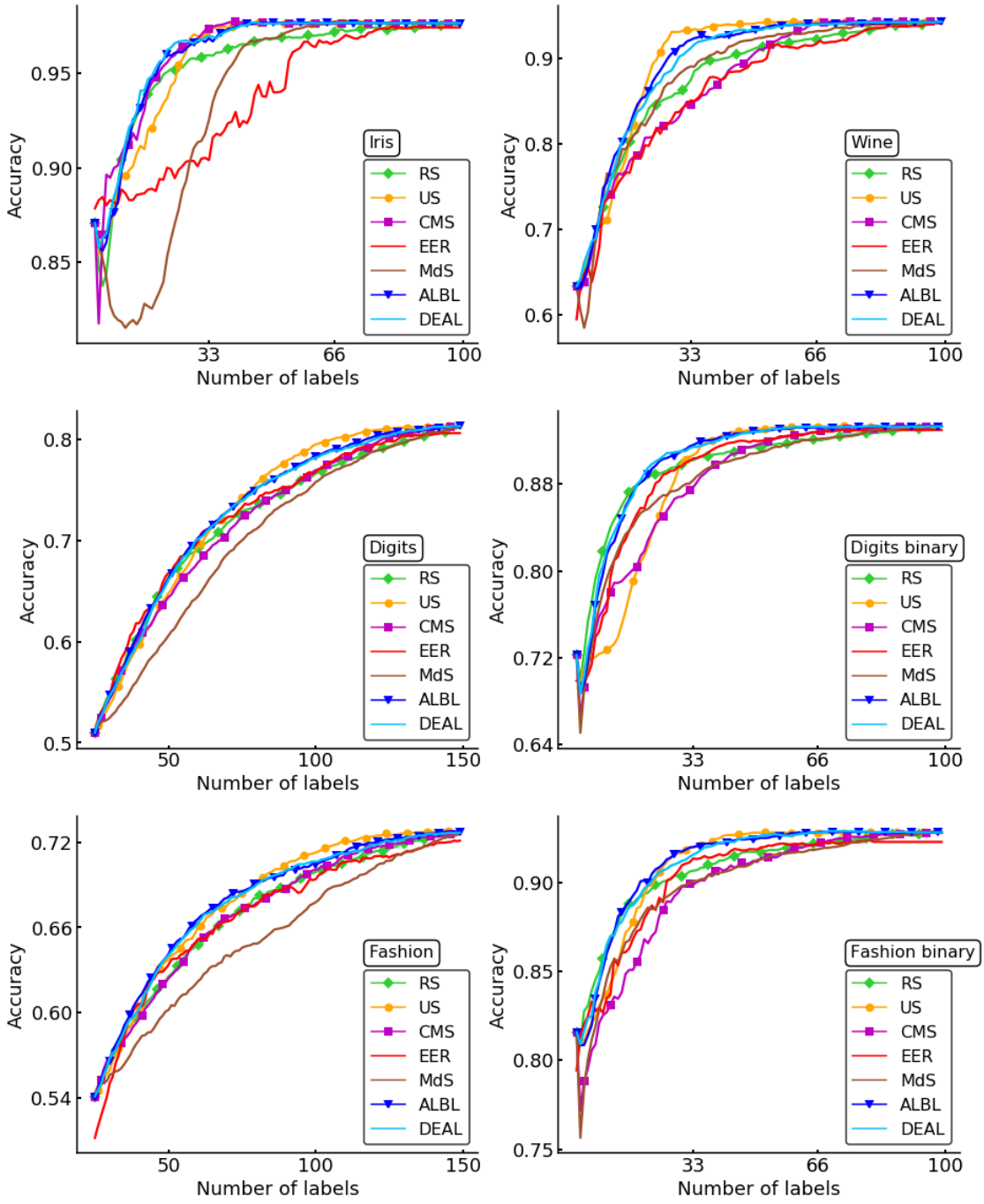


Figure 4.8.: Training a linear SVM to classify six different datasets using seven different strategies with parameters given in the figure's parameter sheet. The plot displays the test accuracy of the classifier versus the number of samples in the training pool. ALBL and DEAL both combine RS and US. Displayed is the mean of 50 runs.

considering it only adds a random query as an alternative. We also put emphasis on how well ALBL and DEAL work. They almost seem to foresee the transition point from RS to US. EER performs comparably to RS. CMS and MdS increase label complexity.

Parameter sheet for figure 4.8 and figure 4.10

Datasets: Iris (1), Wine (2), EMNIST-Digits (3), Digits binary (4), MNIST-Fashion (5), Fashion binary (6)

Dataset Number	1	2	3	4	5	6
Number of samples	150	178	280000	56000	70000	14000
Number of features	4	13	784	784	784	784
Number of classes	3	3	10	2	10	2
Initial number of samples	3	3	25	2	25	2
Initial number of classes	3	3	10	2	10	2
Final number of training samples	100	100	150	100	150	100
Number of test samples	50	78	300	300	300	300
Number of runs for averaging	200	75	75	100	75	100
Number of runs for EER	50	20	20	50	20	50

Classifier:

- SVM with linear kernel function (figure 4.8)
- RFC with 10 estimators (figure 4.10)

Query strategies:

	Parameters
RS	-
US	method = 'entropy', model = main classifier
CMS	space = 'full'
EER	depth = 1, model = main classifier
MdS	goal = 'low', space = 'unlabeled', metric = manhattan
ALBL	query_strategy = {US}, uniform_sampler = True, model = main classifier, T = 97/98/125, δ = 0.1
DEAL	query_strategy = {RS, US}, model = main classifier, γ = 0.1, Δ_T = 10, T = 97/98/125, α = 0.1, β = 10

Now, we have established that pool-based active learning is capable of decreasing label complexity. We try to answer the cost of deploying a query strategy as every query strategy will require significantly more computational effort than RS.
To compare computation time we define

$$\tau(Q, L) \tag{4.2}$$

as the time it takes Q to propose the next sample to label when the labeled pool contains L samples. This allows us to then define two mean times

$$\bar{\tau}_N(Q, L) = \frac{1}{N} \sum_{n=1}^{N} \tau(Q, L) \tag{4.3}$$

and

$$\bar{\bar{\tau}}_N(Q) = \frac{1}{L_{\text{final}} - L_{\text{init}}} \sum_{L=L_{\text{init}}}^{L_{\text{final}}} \bar{\tau}_N(Q, L) \tag{4.4}$$

where L_{final} and L_{init} is the final and initial number of labels. Hence, we can use $\bar{\tau}_N(Q, L)$ to compare how the computation time of query strategy Q changes as the number of labels increases. We use $\bar{\bar{\tau}}_N(Q)$ to compare the computational requirement of different query strategies.
Figure 4.9 displays $\bar{\tau}_{50}$ of different query strategies versus the number of labels using the Iris dataset.
Note that in figure 4.9, and as always, the query strategies merely rearrange a limited training pool (of in this case 100 samples). This explains the (not-exclusive) tendency of query strategies becoming quicker as the number of labels increase, since the unlabeled pool is getting smaller as the query strategies have less options to choose from. This also explains the artifact at the last iteration, so when going from 99 to 100 labels. There is only one unlabeled sample left and therefore, the query strategy has no more choice.
Finally, table 4.1 compares query times averaged over all query iterations of different query strategies on different datasets.
We note two results. First, combining RS, US and CMS by using ALBL or DEAL leads to a significant increase in computational time compared to the summed effort of RS, US and CMS. Secondly, EER is between one to two orders of magnitude ($\approx$ 30-times) slower than all other query strategies. E.g., in case of the EMNIST-Digits dataset the average query iteration of EER requires above one second ($1.4s$).

Overall, it is clear that pool-based active learning is able to decrease label complexity, but with the added difficulty of choosing an appropriate query strategy given the dataset. US performs well on all datasets. We conclude that combining US with an additional query strategy specific to the dataset using ALBL and DEAL seems appropriate. The added computational time this approach requires is negligible.
All results of this section use a linear SVM as a classifier. The next step is to try to duplicate these results using a different classifier.

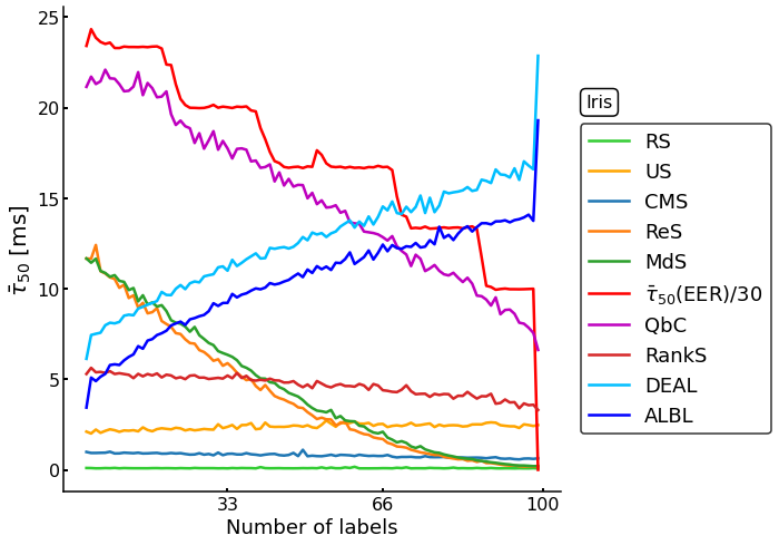


Figure 4.9.: Comparison of computation time of different query strategies during the training process using the Iris dataset and a linear SVM. The y-axis shows $\overline{\tau}_{50}$ (definition eq. (4.3)), the time a query strategy needs to determine the next sample to label, averaged over 50 runs. $\overline{\tau}_{50}$(EER)/30 means that the times for EER are 30 times larger than what is plotted. RankS, ALBL and DEAL all have RS, US and CMS as base query strategies. Computation was performed on a two-core, four-threaded Xeon CPU clocked at 2.2 GHz.

Query strategy →	**RS**	**US**	**CMS**	**ReS**	**MdS**
Dataset ↓ | $\bar{\bar{\tau}}_{10}$ [ms] ↘	**EER**	**QbC**	**RankS**	**DEAL**	**ALBL**
Iris	$1.2 \cdot 10^{-1}$	1.7	$6.6 \cdot 10^{-1}$	1.1	1.2
	$2.2 \cdot 10^{2}$	6.8	2.5	5.2	4.5
Wine	$1.2 \cdot 10^{-1}$	$1.4 \cdot 10^{2}$	$7.1 \cdot 10^{-1}$	1.2	1.2
	$5.1 \cdot 10^{3}$	$1.1 \cdot 10^{2}$	$1.1 \cdot 10^{2}$	$2.2 \cdot 10^{2}$	$2.3 \cdot 10^{2}$
Digits	$2.1 \cdot 10^{-1}$	$3.5 \cdot 10^{1}$	1.4	1.8	1.4
	$1.4 \cdot 10^{3}$	$5.8 \cdot 10^{1}$	$3.7 \cdot 10^{1}$	$7.3 \cdot 10^{1}$	$7.3 \cdot 10^{1}$
Digits binary	$2.2 \cdot 10^{-1}$	$1.3 \cdot 10^{1}$	1.3	1.6	1.5
	$3.1 \cdot 10^{2}$	$2.2 \cdot 10^{1}$	$1.4 \cdot 10^{1}$	$2.6 \cdot 10^{1}$	$2.6 \cdot 10^{1}$
Fashion	$1.9 \cdot 10^{-1}$	$3.1 \cdot 10^{1}$	1.3	1.7	1.4
	$1.4 \cdot 10^{3}$	$5.3 \cdot 10^{1}$	$3.3 \cdot 10^{1}$	$6.6 \cdot 10^{1}$	$6.5 \cdot 10^{1}$
Fashion binary	$2.0 \cdot 10^{-1}$	$1.0 \cdot 10^{1}$	1.3	1.6	1.6
	$3.0 \cdot 10^{2}$	$1.8 \cdot 10^{1}$	$1.2 \cdot 10^{1}$	$2.2 \cdot 10^{1}$	$2.2 \cdot 10^{1}$

Table 4.1.: Comparing computational time (definition eq. (4.4)) of different query strategies on different datasets averaged over the training process for a linear SVM. RankS, ALBL and DEAL all have RS, US and CMS as base query strategies. Computation was performed on a i7-8700 (6-core, 12-threaded, @4.2GHz).

4.2.3. Random forest classifier (RFC)

We repeat the exact calculations of the previous section but with a RFC with ten estimators instead of a linear SVM.
Therefore, the only difference between figure 4.8 and 4.10 and also between the tables 4.1 and 4.2 lies in the classifier choice. The parameters of all involved query strategies are given in the corresponding parameter sheet.

We again go through the results of figure 4.10 by datasets.
In case of the Iris dataset we find multiple query strategies performing very well. E.g., the test accuracy using US peaks with less than 30 samples. Consequently, US results in a decrease of label complexity of $\approx 70\%$. In the previous section, using a linear SVM, EER performed poorly. This is no longer the case using a RFC. Instead, CMS now increases label complexity when using a RFC.
For the Wine dataset US and EER lead to a decrease of label complexity. MdS performs poorly. The remaining query strategies perform comparably or slightly better than RS.
In case of the EMNIST-Digits and MNIST-Fashion datasets all query strategies except for MdS and CMS lead to a small decrease of label complexity. Surprisingly, for the MNIST-Fashion dataset EER performs best.
In both binary datasets US, EER, ALBL and DEAL all decrease label complexity. E.g., in case of the binary Fashion dataset US manages to peak after only 50 labels. Therefore, US reduces label complexity by $\approx 50\%$.
Table 4.2 gives the computation time of every query strategy for the different datasets and averaged over the query iterations. We find similar results as in the previous section. EER is faster on, e.g., the MNIST-Digits dataset than before. This is simply due to the fact that the RFC fits the MNIST-Digits dataset faster than the linear SVM. We can see this by, e.g., comparing the computational time of US on the MNIST-Digits dataset using a linear SVM ($3.5 \cdot 10^1$ms) to using a RFC ($1.1 \cdot 10^1$ms).

As a whole pool-based active learning is able to decrease label complexity in several datasets and using different classifiers. The additional computation time it requires is insignificant except when using EER. With this in mind we finish our investigation of pool-based active learning on toy datasets.

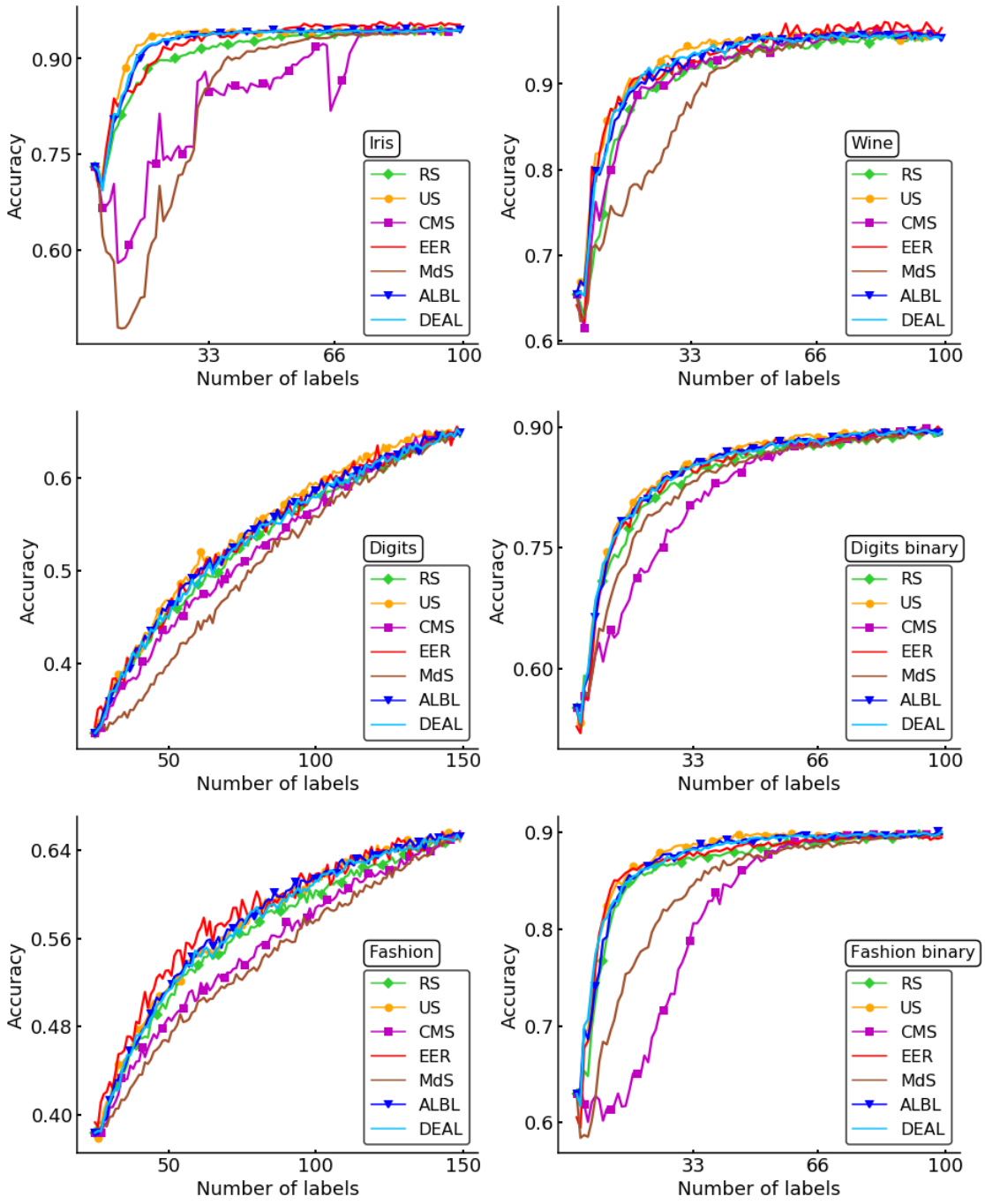


Figure 4.10.: Training a RFC with ten estimators to classify six different datasets using seven different strategies with parameters given in the figure's parameter sheet. The plot displays the test accuracy of the classifier versus the number of samples in the training pool. ALBL and DEAL both combine RS and US. Displayed is the mean of 50 runs.

Query strategy →	**RS**	**US**	**CMS**	**ReS**	**MdS**
Dataset ↓ | $\bar{\bar{\tau}}_{10}$ [ms] ↘	**EER**	**QbC**	**RankS**	**DEAL**	**ALBL**
Iris	$1.2 \cdot 10^{-1}$	$1.0 \cdot 10^{1}$	$6.5 \cdot 10^{-1}$	1.2	1.4
	$4.0 \cdot 10^{2}$	$6.0 \cdot 10^{1}$	$1.1 \cdot 10^{1}$	$3.9 \cdot 10^{1}$	$3.9 \cdot 10^{1}$
Wine	$1.8 \cdot 10^{-1}$	$1.0 \cdot 10^{1}$	$8.4 \cdot 10^{-1}$	1.5	1.4
	$4.0 \cdot 10^{2}$	$6.1 \cdot 10^{1}$	$1.1 \cdot 10^{1}$	$4.0 \cdot 10^{1}$	$3.9 \cdot 10^{1}$
Digits	$1.8 \cdot 10^{-1}$	$1.1 \cdot 10^{1}$	1.3	1.7	1.4
	$8.9 \cdot 10^{2}$	$6.1 \cdot 10^{1}$	$1.4 \cdot 10^{1}$	$4.5 \cdot 10^{1}$	$4.4 \cdot 10^{1}$
Digits binary	$1.7 \cdot 10^{-1}$	$1.0 \cdot 10^{1}$	1.2	1.6	1.3
	$3.6 \cdot 10^{2}$	$6.0 \cdot 10^{1}$	$1.2 \cdot 10^{1}$	$4.2 \cdot 10^{1}$	$4.1 \cdot 10^{1}$
Fashion	$1.9 \cdot 10^{-1}$	$1.2 \cdot 10^{1}$	1.4	1.8	1.4
	$8.9 \cdot 10^{2}$	$6.1 \cdot 10^{1}$	$1.4 \cdot 10^{1}$	$4.6 \cdot 10^{1}$	$4.4 \cdot 10^{1}$
Fashion binary	$1.7 \cdot 10^{-1}$	$1.1 \cdot 10^{1}$	1.4	1.5	1.5
	$3.6 \cdot 10^{2}$	$6.0 \cdot 10^{1}$	$1.3 \cdot 10^{1}$	$4.1 \cdot 10^{1}$	$4.1 \cdot 10^{1}$

Table 4.2.: Comparing computational time (definition eq. (4.4)) of different query strategies on different datasets averaged over the training process for a RFC with ten trees. RankS, ALBL and DEAL all have RS, US and CMS as base query strategies. Computation was performed on a i7-8700 (6-core, 12-threaded, @4.2GHz).

4.3. On-line active learning

We switch from the pool-based approach to on-line active learning. This means we apply the query strategies of section 2.2.2 to the six toy datasets of section 4.1. We compare yet again two classifiers - a linear SVM and a RFC with ten estimators.

4.3.1. Support vector machine (SVM)

We chose a linear kernel function for the SVM. Remember that on-line active learning is more label efficient by occasionally choosing that a sample should not be labeled whereas passive learning (RS) chooses to label every sample. Hence, the goal is to maintain a similar test accuracy as passive learning while requesting less labels.
Therefore, to evaluate the effectiveness of the different query strategies we have to consider two plots simultaneously. The first plot displays the test accuracy versus the number of queries. The second plot shows the number of requested labels as a function of the number of queries. The number of queries can also be interpreted as a time axis if we assume that unlabeled samples are created at a constant rate. The occurrence of every unlabeled sample corresponds to a query, i.e., a decision whether or not the sample should be labeled. In the second plot RS will show up as a line with slope one, i.e., RS decides to label every sample such that every query results in a requested label.
For all six dataset and five query strategies figure 4.13 and 4.14 display the test accuracy and number of requested labels as a function of the number of queries.

The results can be interpreted in two ways.
Equal cost/labels:
As before we refer to the plot of test accuracy versus number of queries as the first plot and to the plot of the number of requested labels versus number of queries as the second plot.
We can compare two query strategies in on-line active learning in the following manner. In the second plot choose a value for the number of requested labels, check how many number of queries both query strategies require to reach the number of requested labels. In the first plot see which accuracy both query strategies achieve at that number of queries. This compares the accuracy of both query strategies at equal number of labels/cost.
This procedure is also shown in figure 4.11.

Equal test accuracy:
In the first plot choose a test accuracy and see how many queries both query strategies need to achieve the chosen test accuracy. Then, in the second plot compare how many labels both query strategies have requested at the corresponding number of queries. This directly compares label complexity. The procedure is also shown in figure 4.12.

With this in mind we take a closer look at the figures 4.13 and 4.14. Overall, we find on-line active learning showing good performance of the involved algorithms in the

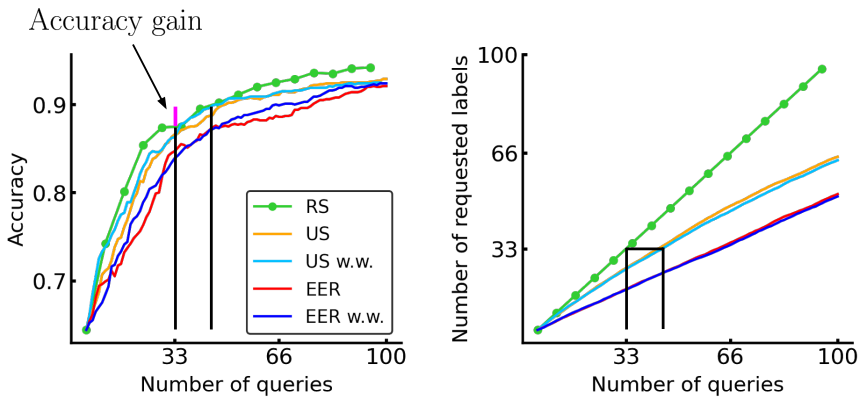


Figure 4.11.: Procedure used to compare two query strategies in on-line active learning using a linear SVM and the Wine dataset. Here, we estimate the accuracy gain by using US w.w. over RS at equal number of labels.

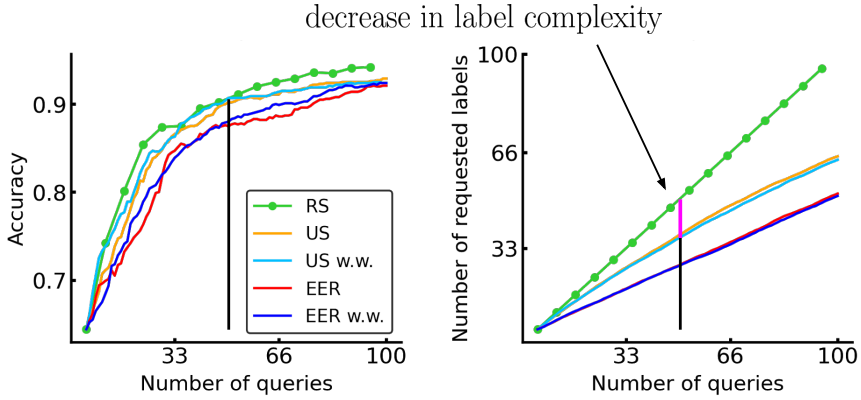


Figure 4.12.: Procedure used to compare two query strategies in on-line active learning using a linear SVM and the Wine dataset. Here, we estimate the decrease in label complexity by using US w.w. over RS at equal accuracy.

less complex datasets Iris, Wine, binary Digits and binary Fashion but struggling for the EMNIST-Digits and MNIST-Fashion datasets. In most cases US outperforms EER while the addition of weights does not seem to play a significant role either way.

4.3.2. Random forest classifier (RFC)

We repeat the same calculations as in the previous chapter but chose a RFC with ten estimators as our classifier. We average over 50 runs resulting in figure 4.15 and 4.16. The only difference between the figure 4.13 and 4.15 and also between figure 4.14 and 4.16 lies in the choice of classifier.

We find identical results as in the previous chapter. On-line active learning is able to decrease label complexity for the Iris, Wine, binary Digits and binary Fashion dataset. E.g., for the Wine dataset after ≈ 30 queries US w.w. is able to match the accuracy of RS while having only requested half of the labels. On-line active learning performs subpar in the EMNIST-Digits and MNIST-Fashion datasets.

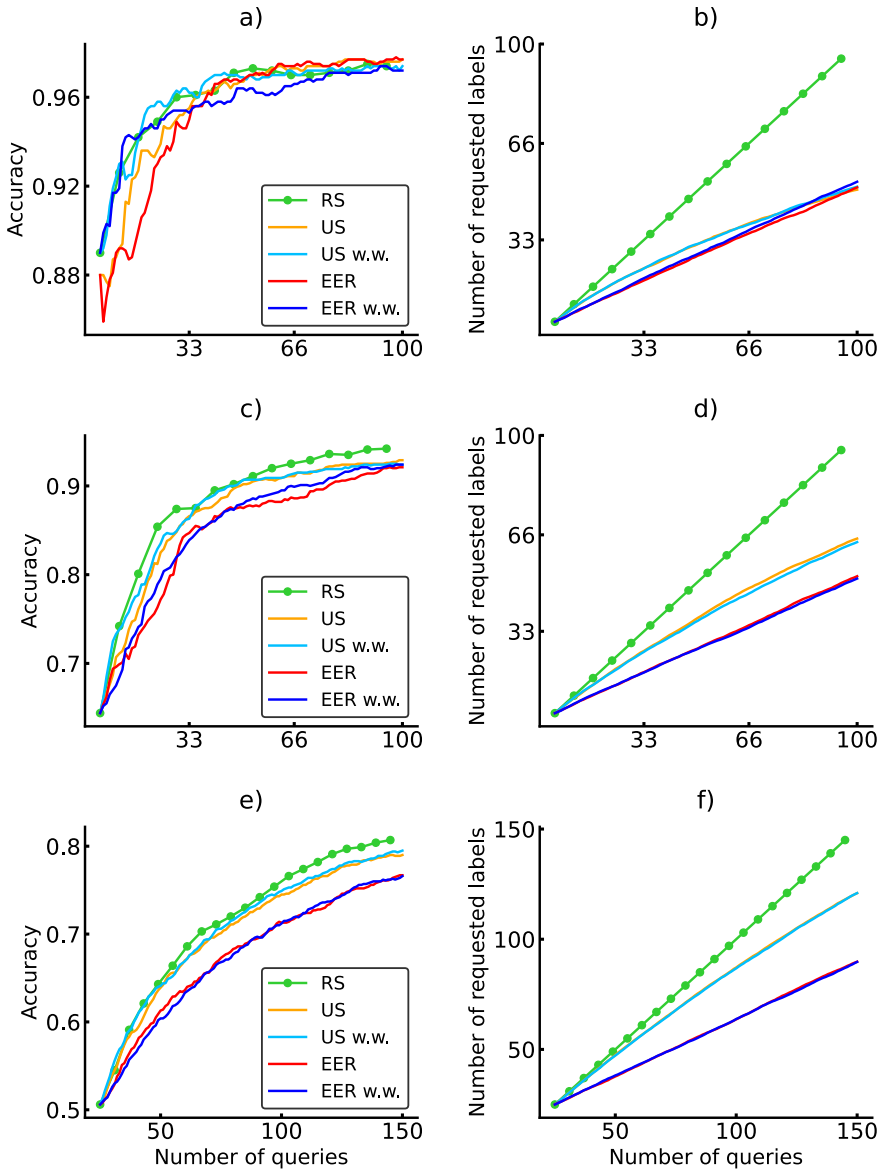


Figure 4.13.: Performance comparison of on-line active learning using different strategies and datasets, the trained model is a SVM with linear kernel. Two subplots should be interpreted together, e.g., for the Iris dataset subplot a) shows the test accuracy as a function of the number of queries and subplot b) displays the number of labels the strategy has already requested as a function of the number of queries. The goal of an efficient strategy is to maintain a similar accuracy as RS, while requesting less labels. Similarly, c) and d) is for the Wine, and e) and f) for the EMNIST-Digits dataset. All plots are averaged over 50 runs.

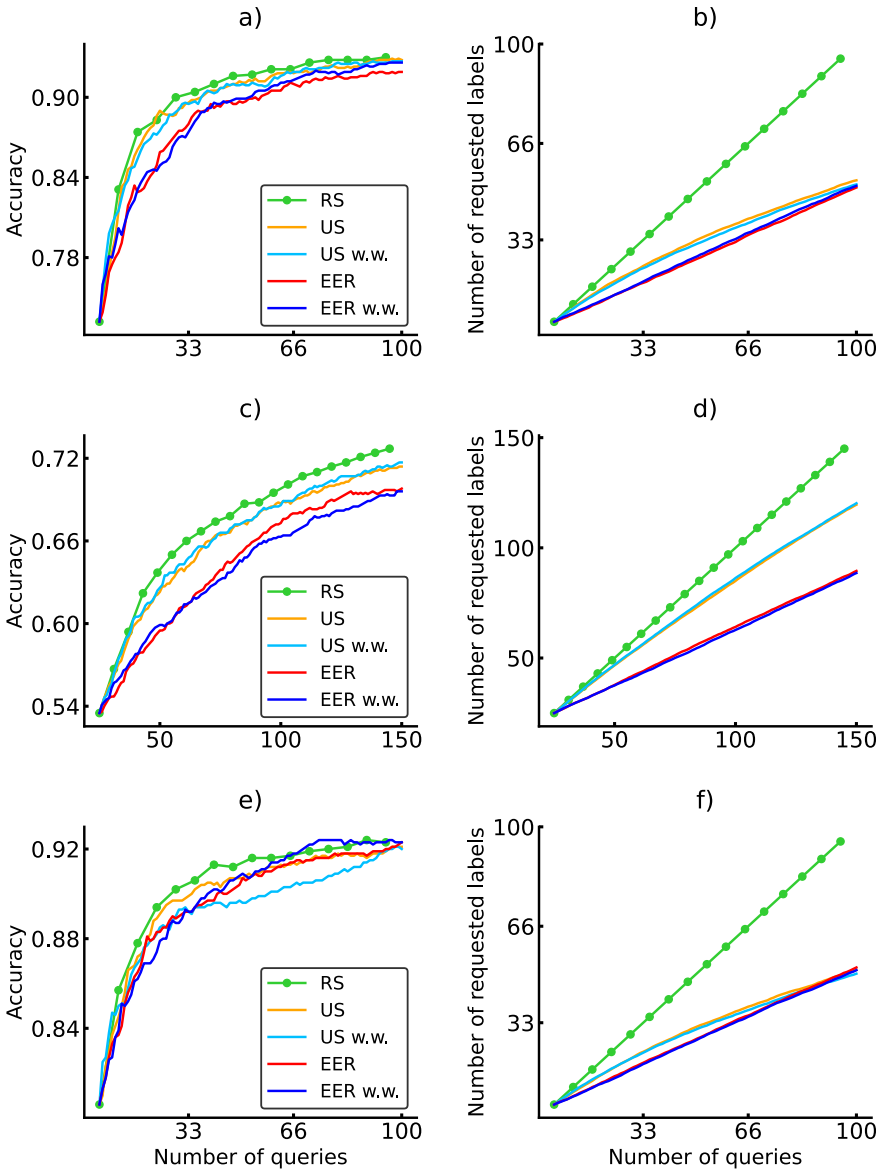


Figure 4.14.: Performance comparison of on-line active learning using different strategies and datasets, the trained model is a SVM with linear kernel. Two subplots should be interpreted together, e.g., for the Digits binary dataset subplot a) shows the test accuracy as a function of the number of queries and subplot b) displays the number of labels the strategy has already requested as a function of the number of queries. The goal of an efficient strategy is to maintain a similar accuracy as RS, while requesting less labels. Similarly, c) and d) is for the MNIST-Fashion, and e) and f) for the Fashion binary dataset. All plots are averaged over 50 runs.

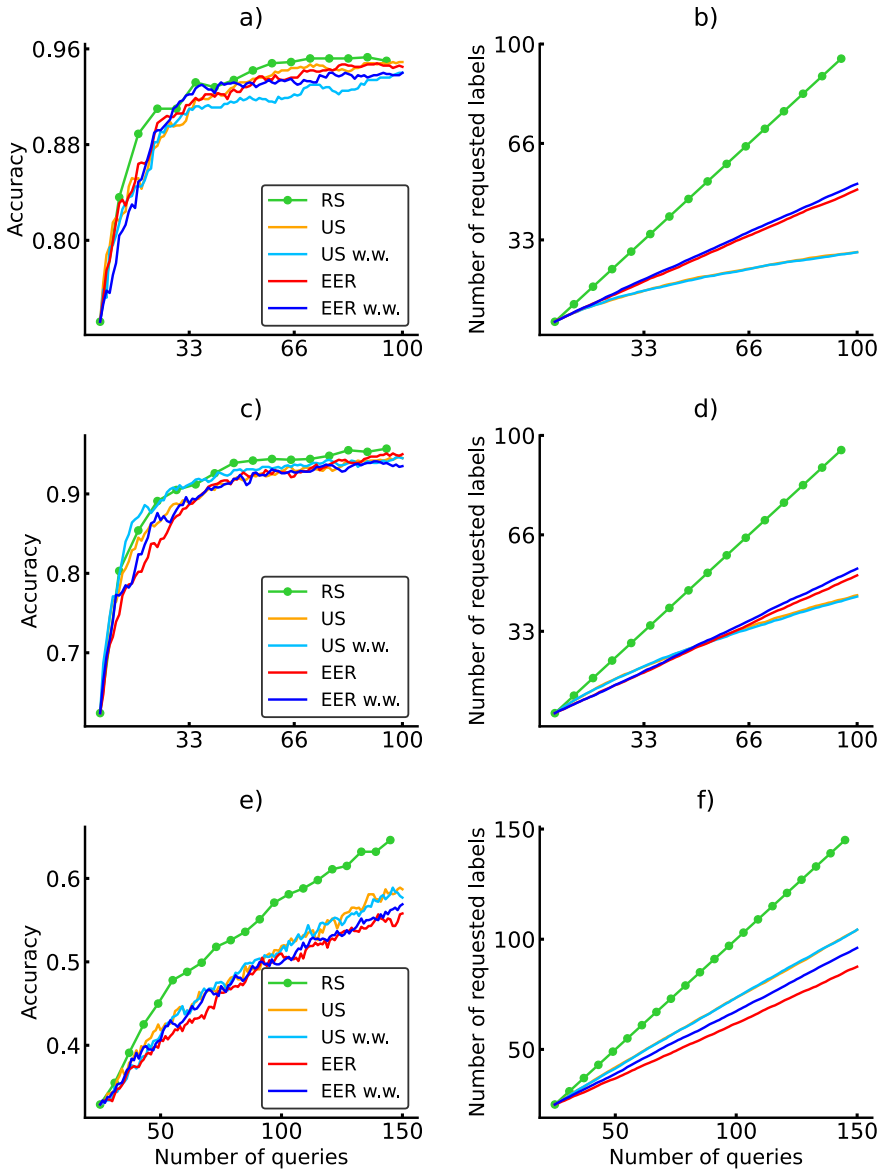


Figure 4.15.: Performance comparison of on-line active learning using different strategies and datasets, the trained model is a RFC with ten estimators. Two subplots should be interpreted together, e.g., for the Iris dataset subplot a) shows the test accuracy as a function of the number of queries and subplot b) displays the number of labels the strategy has already requested as a function of the number of queries. The goal of an efficient strategy is to maintain a similar accuracy as RS, while requesting less labels. Similarly, c) and d) is for the Wine, and e) and f) for the EMNIST-Digits dataset. All plots are averaged over 50 runs.

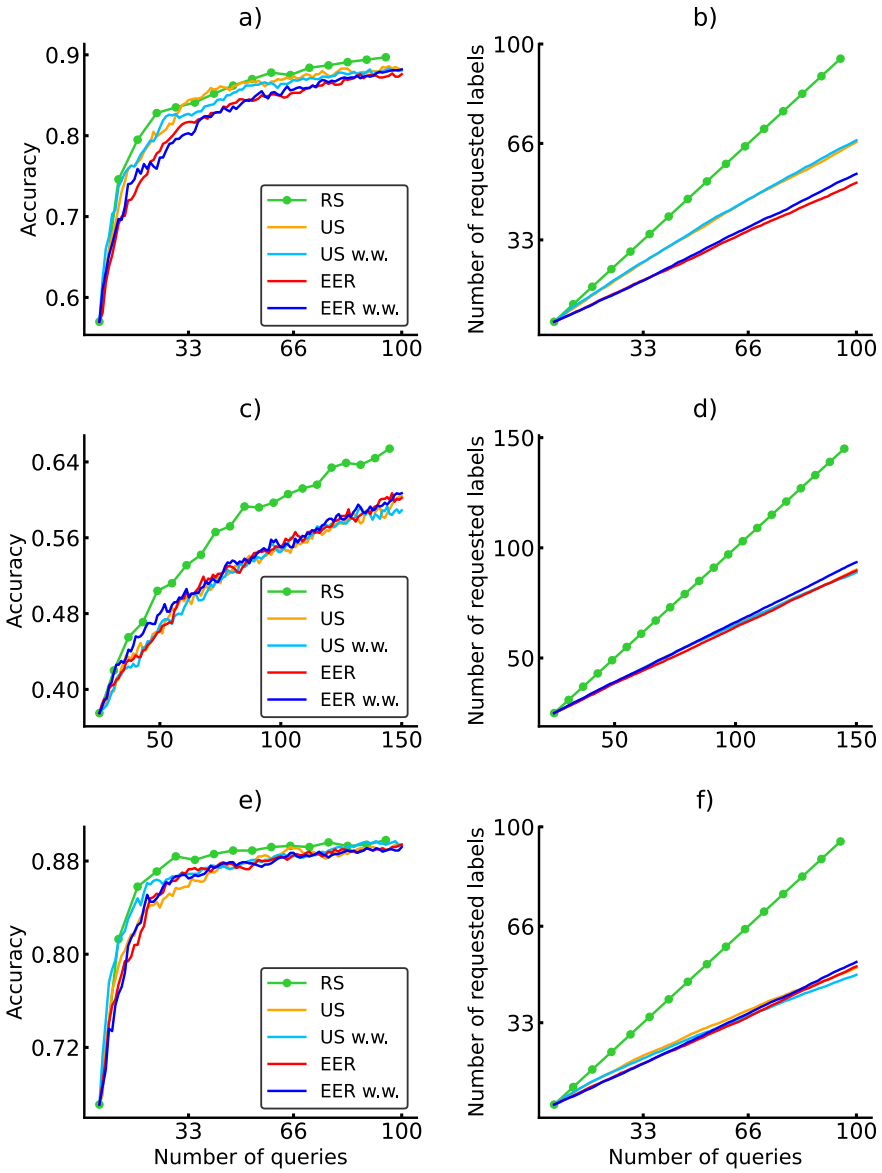


Figure 4.16.: Performance comparison of on-line active learning using different strategies and datasets, the trained model is a RFC with ten estimators. Two subplots should be interpreted together, e.g., for the Digits binary dataset subplot a) shows the test accuracy as a function of the number of queries and subplot b) displays the number of labels the strategy has already requested as a function of the number of queries. The goal of an efficient strategy is to maintain a similar accuracy as RS, while requesting less labels. Similarly, c) and d) is for the MNIST-Fashion, and e) and f) for the Fashion binary dataset. All plots are averaged over 50 runs.

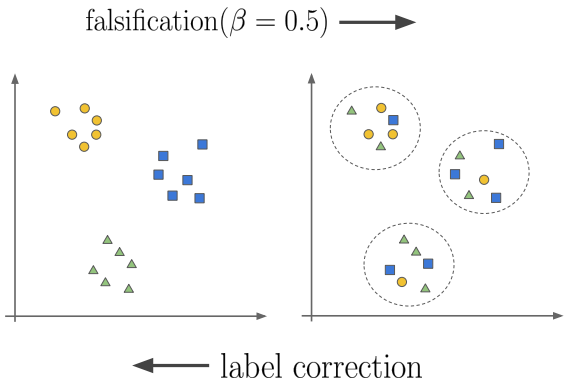


Figure 4.17.: Demonstrating the falsification step on toy datasets consisting of three classes separated in three clusters. Label correction inverts the falsification process by introducing label homogeneity in the clusters. The clusters are indicated as circles.

4.4. Label correction

This section is dedicated to applying the algorithms outlined in section 3.5 to toy datasets. Obviously the toy datasets are certainly very well labeled such that there is nothing to correct. In order to proceed we artificially falsify a percentage of every dataset and then check to which degree we can invert this process and retrieve the original dataset.
Let's start by talking about how we falsify the toy data. We draw randomly a fraction β samples out of the dataset. For each class $z = 1 \rightarrow Z$, on average βN_z instances (N_z is the number of instances of class z) are randomly mislabeled to one of the other $(Z-1)$ classes. Among the βN_z instances, the number of samples labeled to class z' is proportional to $N_{z'}$. Following this procedure keeps the class distribution/ratio of the mislabeled data close to the one of the original data. This is a consequence of assuming that mislabelings have a random nature – each instance has an equal chance to be mislabeled. Based on this assumption, we can infer that the class distribution for a dataset with mislabeling should reflect the one without mislabeling. We adopt this procedure from the following paper [36].
The falsification step and label correction as its inversion is displayed in the toy example in figure 4.17. This should also give an intuition why correct labels are recoverable beyond a falsification degree ≥ 0.5, i.e., when there are as many or more incorrect than correct labels. All label correction methods work by dividing data space in regions and introducing homogeneity in those regions. In the example, a successful label correction method divides the data in three clusters (indicated by circles) and introduces homogeneity in each cluster *separately*.

To quantify the effectiveness of the different correction algorithms we define the correc-

tion factor $\mathcal{C}$ as

$$\mathcal{C}(\beta) \coloneqq \frac{N_c(\beta) - (1-\beta)N}{\beta N} \tag{4.5}$$

where N is the number of samples, β is the mislabeled fraction and N_c is the number of samples with correct label assignment *after* label correction, which obviously depends on β. This means that $\mathcal{C} = 1$ implies that all labels are corrected whereas $\mathcal{C} = 0$ corresponds to as many correct labels as before correction.
Of course this will not be applicable if we never have the correct labels to begin with as it will be the case for the real-world data in the next chapter. Then, an alternative method of gauging the correction algorithms effectiveness would be to train a classifier using the original labels and the corrected labels and compare its test accuracies. This still requires us to have a perfectly labeled (smaller) pool to compute test accuracies. Fortunately, this is exactly the setup of the subsequent chapter on real-world data.

To compare test accuracies before and after correction we use the simplest nearest neighbour classifier, i.e., a K-NN classifier with $K = 1$. It predicts a test sample's label as the label of the closest training sample. To get accurate estimates we use 10-fold cross validation to compute accuracies on data which the classifier has not used for training (definition of test accuracy).
In this context this means we split the entire dataset into ten equally large subsets. Without loss of generality we pick the first subset as our test data and the remaining subsets are training data. Then, we artificially falsify a fraction β of the training data (the nine remaining subsets) and either apply a label correction algorithm or use the data directly to train our classifier. Then, we use the trained classifier to compute the prediction accuracy on the test subset. We average over the ten different choices of test subset. This is again in complete analogy to [36].

Now, we are able to apply the four different algorithms for label correction NNC, ADE, CC and BCC namely to the Iris, Wine and EMNIST-Digits dataset.

4.4.1. Nearest neighbour correction (NNC)

We use algorithm 20 with $K = 8$ and confidence $p = 0.5$ on three artificially falsified datasets corresponding to Iris, Wine and EMNIST-Digits with different levels of falsification β.
The resulting correction factor (definition eq. (4.5)) is given in figure 4.18 and the test accuracy of a 1-NN classifier after training on the falsified and on the corrected data is given in figure 4.19.

The novel algorithm NNC is a strong performer. In figure 4.18 we see that it is able to correct a large fraction of falsified data in case of the Iris and EMNIST-Digits dataset. For the EMNIST-Digits dataset even at a falsification of $\beta = 0.5$ NNC manages to almost completely retrieve the original labels. This is due to the fact that EMNIST-Digits has ten classes and the assumption that mislabelings have a random nature.
In figure 4.18 NNC struggles for the Wine dataset and small percentages of false labels β.

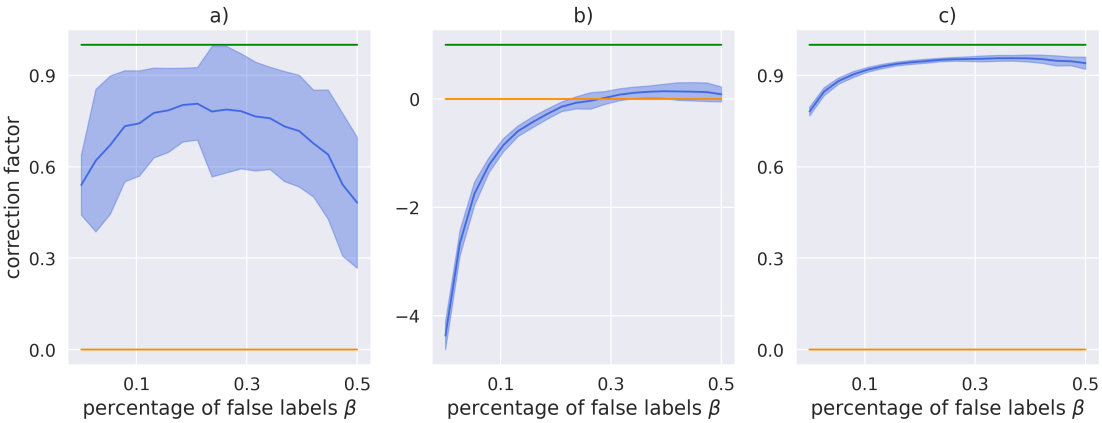


Figure 4.18.: The correction factor (see eq. (4.5)) after applying NNC on falsified Iris a), Wine b) and EMNIST-Digits c) dataset as a function of the fraction of incorrect labels β. A correction factor of one is equivalent to all incorrect labels are corrected whereas a correction factor of zero implies that the correction algorithm has no positive (or negative) impact. Plotted is the mean of 20 runs, ± one standard deviation.

Regardless, in figure 4.19 we can see that NNC still manages to increase the test accuracy of a 1-NN classifier significantly, even in the case of the Wine dataset. This is especially interesting around $\beta \approx 0.3$, we see a correction factor of zero but a huge improvement in the test accuracy of a 1-NN classifier. It is crucial to note that a correction factor of zero only means that there are as many incorrect labels after correction than before. But only the amount of incorrect labels has to be equal - not the *actual* samples. We conclude that NNC reassigns labels in a more efficient way while conserving an equal number of incorrect labels. To do so, it also has to change correct labels into incorrect ones.
There is small improvement in the test accuracy in case of the Iris dataset and no false labels ($\beta = 0$), implying that the Iris dataset may not be perfectly labeled to begin with. Finally, we note that in figure 4.19 it is quite impressive how well NNC manages to maintain a high test accuracy up till large values of falsification.

4.4.2. Automatic data enhancement (ADE)

We apply algorithm 21 with $N_{\text{epoch}} = 50, N_{\text{era}} = 50, \eta_{\boldsymbol{\omega}} = 10^{-5}, \eta_{\boldsymbol{p}} = 10^{-3}, \epsilon = 10^{-5}$ on three artificially falsified datasets corresponding to Iris, Wine and EMNIST-Digits with different levels of falsification β.
The resulting correction factor (definition eq. (4.5)) is given in figure 4.20 and the test accuracy of a 1-NN classifier after training on the falsified and on the corrected data is given in figure 4.21.

The results suggest that ADE only proves effective in case of the Iris dataset. But this is

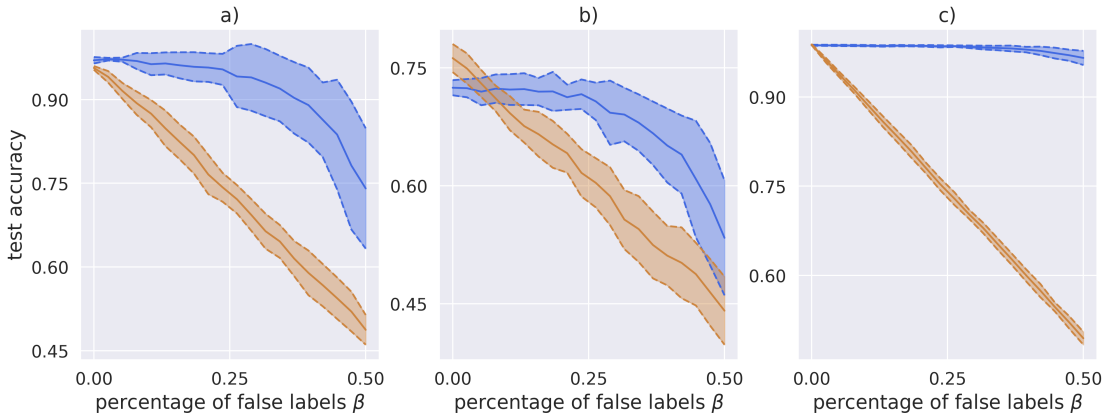


Figure 4.19.: Test accuracy of a 1-NN classifier trained on falsified data (orange curve) and on falsified data after applying NNC (blue curve) as a function of the fraction of incorrect labels β. Datasets are Iris a), Wine b) and EMNIST-Digits c) and 10-fold CV is used to estimate test accuracies. Plotted is the mean of 20 runs, ± one standard deviation.

misleading since ADE heavily depends on hyperparameters and the above chosen parameters are carefully chosen for the Iris dataset. Unfortunately, choosing well calibrated hyperparameters for the Wine and EMNIST-Digits dataset is non-trivial. This hyperparameter complexity turns out to be ADE's biggest downfall as it makes it difficult to present an out-of-the-box working algorithm which is always desirable.
Regardless, in case of the Iris dataset (with well chosen hyperparameters) the results of ADE are positive.

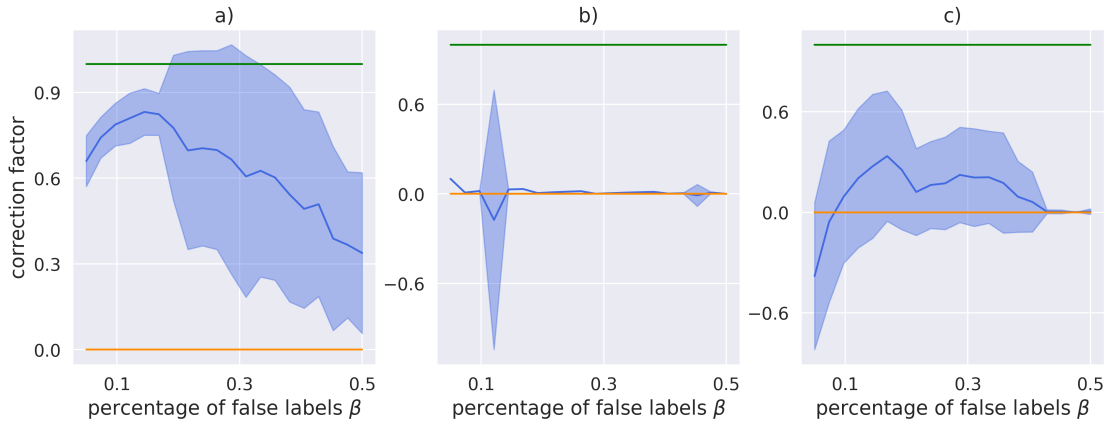


Figure 4.20.: The correction factor (see eq. (4.5)) after applying ADE on falsified Iris a), Wine b) and EMNIST-Digits c) dataset as a function of the fraction of incorrect labels β. A correction factor of one is equivalent to all incorrect labels are corrected whereas a correction factor of zero implies that the correction algorithm has no positive (or negative) impact. ADE performs poorly on Wine and EMNIST-Digits, this is due to difficult to choose hyperparameters. Plotted is the mean of 20 runs, ± one standard deviation.

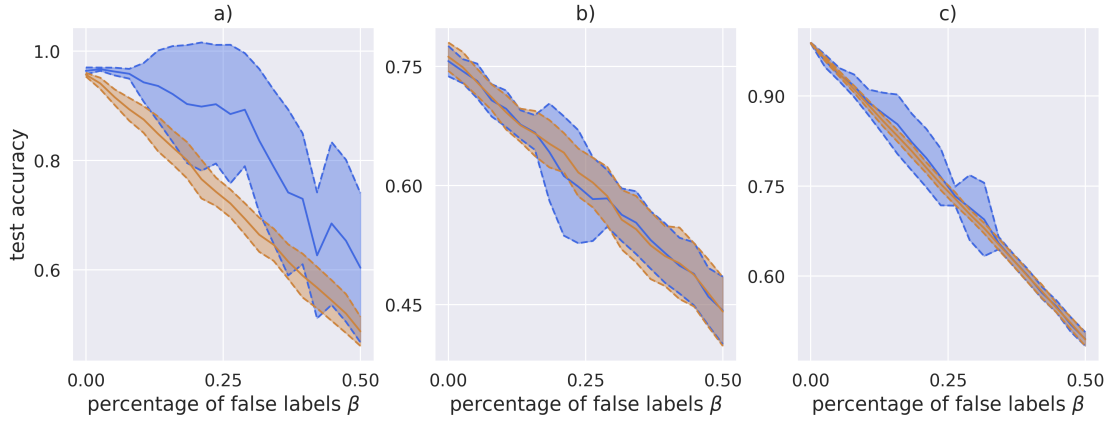


Figure 4.21.: Test accuracy of a 1-NN classifier trained on falsified data (orange curve) and on falsified data after applying ADE (blue curve) as a function of the fraction of incorrect labels β. Datasets are Iris a), Wine b) and EMNIST-Digits c) and 10-fold CV is used to estimate test accuracies. Plotted is the mean of 20 runs, ± one standard deviation.

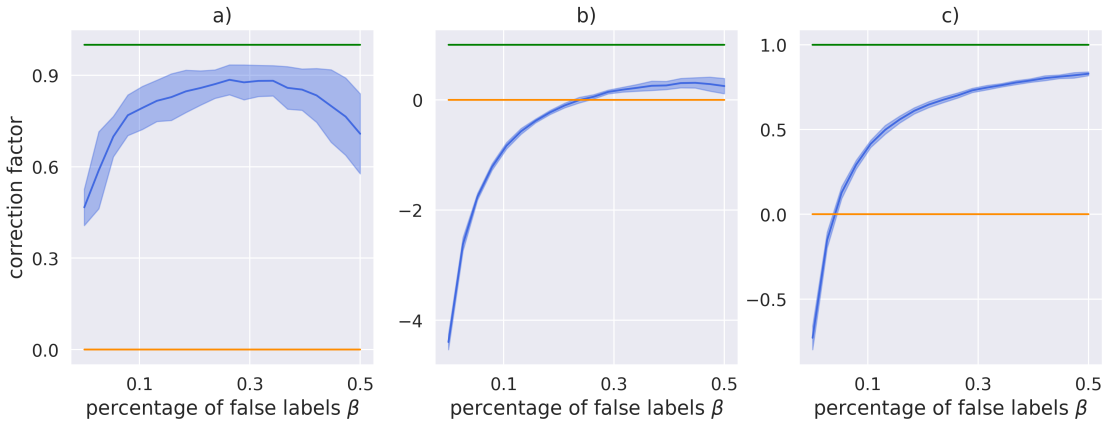


Figure 4.22.: The correction factor (see eq. (4.5)) after applying CC on falsified Iris a), Wine b) and EMNIST-Digits c) dataset as a function of the fraction of incorrect labels β. A correction factor of one is equivalent to all incorrect labels are corrected whereas a correction factor of zero implies that the correction algorithm has no positive (or negative) impact. Plotted is the mean of 20 runs, ± one standard deviation.

4.4.3. Cluster correction (CC)

Next, we continue by applying algorithm 22 with number of clusterings $A = 25$ on three artificially falsified datasets corresponding to Iris, Wine and EMNIST-Digits with different levels of falsification β.
The resulting correction factor (definition eq. (4.5)) is given in figure 4.22 and the test accuracy of a 1-NN classifier after training on the falsified and on the corrected data is given in figure 4.23.

CC also proves, next to NNC, as a strong performer. In figure 4.23 we can see that CC manages to maintain a high test accuracy for all three datasets.
It should be noted that, in case of the Wine and EMNIST-Digits dataset and a low percentage of false labels it proves counter-productive and slightly decreases test accuracy of the 1-NN classifier.
CC also shows similar behaviour to NNC when we look at the Wine dataset and a falsification of $\beta \approx 0.25$. Despite a correction factor of zero, CC significantly increases the test accuracy of a 1-NN classifier. This again suggests that CC also turns correct into incorrect labels.
Further, CC also manages to slightly increase the test accuracy in case of the Iris dataset and no false labels and thereby implies false labels in the original dataset.

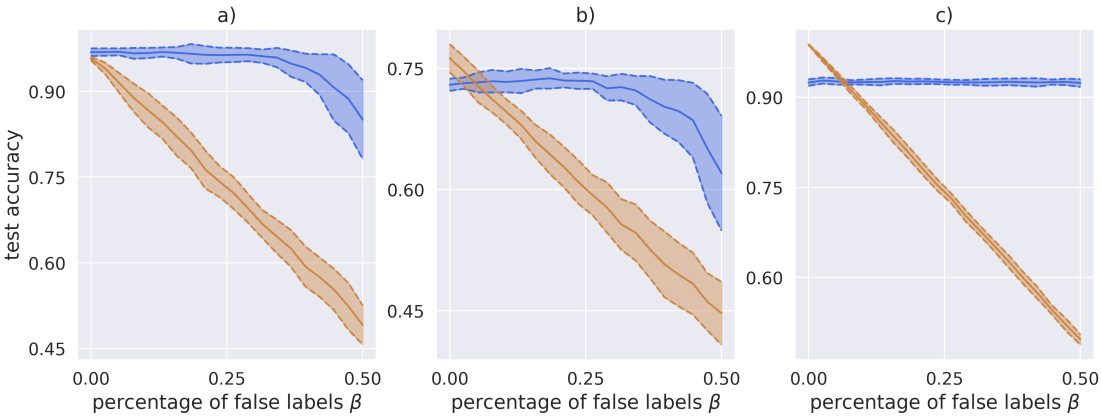


Figure 4.23.: Test accuracy of a 1-NN classifier trained on falsified data (orange curve) and on falsified data after applying CC (blue curve) as a function of the fraction of incorrect labels β. Datasets are Iris a), Wine b) and EMNIST-Digits c) and 10-fold CV is used to estimate test accuracies. Plotted is the mean of 20 runs, ± one standard deviation.

4.4.4. Binary cluster correction (BCC)

Finally, we use the algorithm of section 3.5.4 for label correction (BCC does not have any hyperparameters) on three artificially falsified datasets corresponding to Iris, Wine and EMNIST-Digits with different levels of falsification β.
The resulting correction factor (definition eq. (4.5)) is given in figure 4.24 and the test accuracy of a 1-NN classifier after training on the falsified and on the corrected data is given in figure 4.25.

BCC shows decent performance but is inferior to both NNC and CC. Still, it results in a substantial improvement in test accuracy for all three datasets and a higher degree of falsification ($\beta \geq 0.2$). In case of the Wine and EMNIST-Digits dataset and a low degree of falsification BCC disimproves the test accuracy, especially for the EMNIST-Digits dataset.

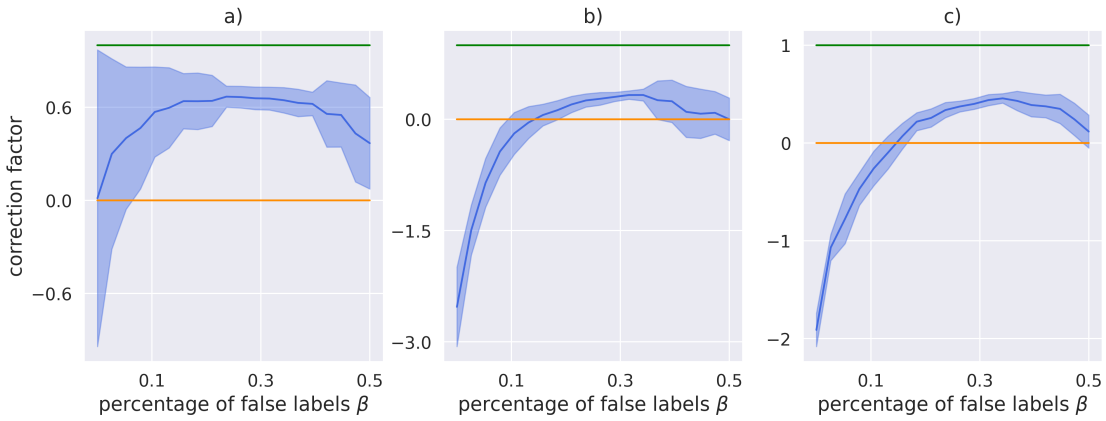


Figure 4.24.: The correction factor (see eq. (4.5)) after applying BCC on falsified Iris a), Wine b) and EMNIST-Digits c) dataset as a function of the fraction of incorrect labels β. A correction factor of one is equivalent to all incorrect labels are corrected whereas a correction factor of zero implies that the correction algorithm has no positive (or negative) impact. Plotted is the mean of 20 runs, ± one standard deviation.

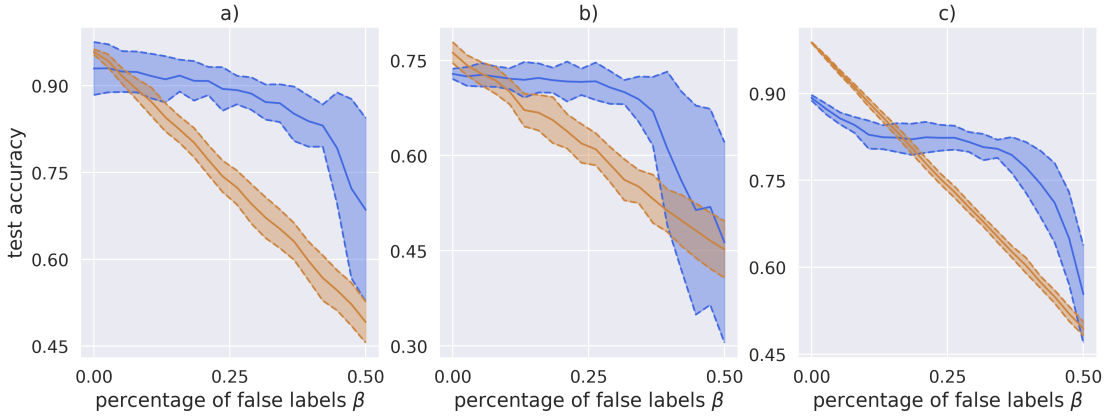


Figure 4.25.: Test accuracy of a 1-NN classifier trained on falsified data (orange curve) and on falsified data after applying BCC (blue curve) as a function of the fraction of incorrect labels β. Datasets are Iris a), Wine b) and EMNIST-Digits c) and 10-fold CV is used to estimate test accuracies. Plotted is the mean of 20 runs, ± one standard deviation.

5. Results on real-world data

In the previous chapter we have seen active learning being applied to several toy/benchmark datasets. While this can prove a working concept, it is still important to replicate similar results on real-world data. Therefore, we will start by explaining the problem formulation which we want to solve using some specific data and also talk about the data encoding. Finally, in this scenario we are also provided with an auxiliary data source which will allow us to apply the theory from chapter 3.

5.1. Real-world data

The problem formulation and data originate in a bottle filling plant. There, several machines work together in a chain to form a production line. Every machine fulfills a certain task such as cleaning, filling or labeling. The filling machine in this production line is of special importance. The filler should not have too many or too long stops for ensuring product quality, i.e., it should always be working, so filling bottles. To ensure that, an artificial bottleneck is created at the filling machine. This means all machines before and after the filling machine have a higher pass-through bottle performance than the filler, while the filling machine runs at full throttle. This is called a V-shaped production line, since the pass-through capability of a machine as a function of the machines ordered along the bottle stream, follows a V-shape. By doing so smaller fluctuations of the bottle current happening away from the filling machine never disturb its flow.
Still, sometimes the filling machine will enter a non-productive state. The task is to predict the machine where the failure, leading to the filler's non-productive state, originated. For that we are provided with a time series of machine states *before* (due to causality) the filler entered the non-productive state, including the moment of the filler stop. The time series is a record of every machine's current state at several time stamps (before the incident at the filler). There are four machine states - productive, lack (too little bottles before the machine), tailback (too many bottles after the machine) and failure. To understand this better look at the artificial example in figure 5.1. Here the root cause for the non-productive state of the lead machine 6, is the previous failure of machine 4. Therefore, the label for this time series would be machine 4.
Unfortunately, machine-states captured in a time series do regularly not paint a clear picture, making the labeling often non-obvious, see figure 5.2 for an example.
Finally we are given two separate datasets - the primary pool, labeled by a human and assumed to be perfectly labeled and the auxiliary pool, labeled by a hard-coded logic with unknown labeling accuracy.

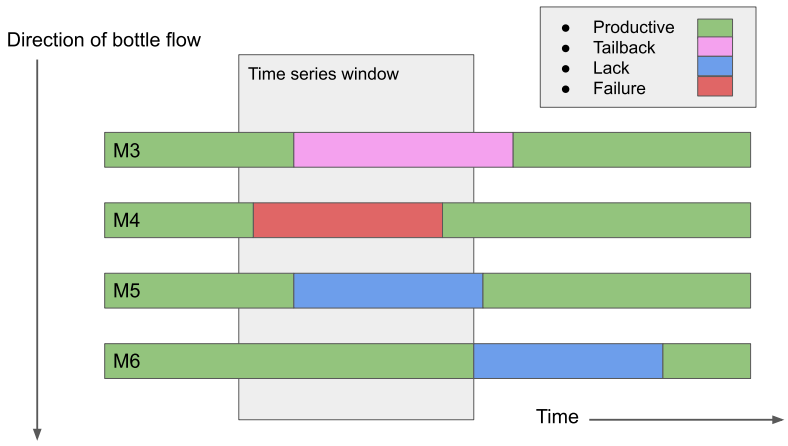


Figure 5.1.: Artificial example of machine states that lead to a time series with label machine 4. Machine 4 experiences a failure, this leads to a retarded tailback at machine 3 and retarded lacks in machine 5 and 6. The time series window is positioned right before the head machine 6 enters a non-productive state.

Having understood the real-world data we continue by discussing the framework of active learning in the context of our real-world data and finally use the methods from chapter 3 on the auxiliary pool.

Characteristic features of real-world data:

Dataset name	primary data	auxiliary data
Number of samples	342	6703
Number of features	36	36
Number of classes	9	10

5.2. Active learning

When the idea of a master thesis that treats active learning originated, the ulterior motive was to ultimately implement active learning in a colleague's project (the real-world data) to decrease the labeling efforts.
The most obvious realization of active learning in the classification setup our real-world data poses, is to maintain a large unlabeled pool of past errors that is updated by a sample whenever a machine failure occurs. Then, occasionally the expert decides to label some samples and the query strategy proposes data points to label from the large pool. Since a time series of machine states can be highly cryptic (e.g. figure 5.2), this poses the obvious problem of having to label machine failures that happened in the far

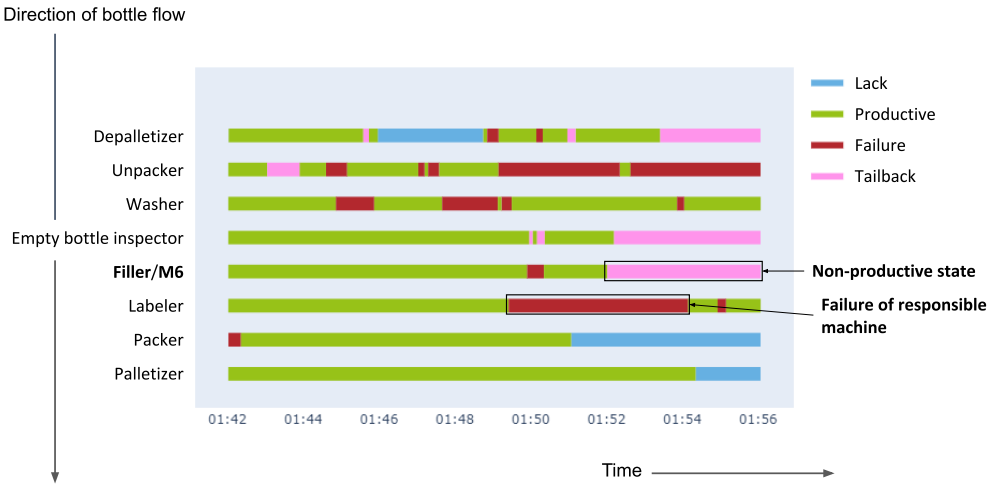


Figure 5.2.: Real-world example of machine states that lead to a time series with label machine 7/Labeler.

past or may have not even been observed during the expert's shift.
A simple solution is to keep the unlabeled pool small by only saving the failures that occurred during a shift and at the end of this shift the operator/expert labels a fraction of this smaller pool. Afterwards, all remaining unlabeled samples get discarded and then the next shift with a different expert starts. This solves both previously mentioned problems by guaranteeing that data points are at most one shift duration old and have obviously happened on the expert's watch.
We now investigate this framework regarding what value for the fraction of number of samples that get labeled gives the best compromise between accuracy and labeling effort. This assumes that we have already found a query strategy that works efficiently on the given data, so let's take one step back and start from there.

As our classifier we chose a SVM with polynomial kernel of degree three. The two query strategies that emerge as the best performing are US and NNC. Their performance is illustrated in figure 5.3.
NNC outperforms RS and US early on while US outperforms RS and NNC later. By combining both using DEAL we are able to heavily decrease label complexity at all stages. E.g., DEAL peaks at ≈ 60 labels. Thereby, acquired labels past that point are a waste as they don't contribute any additional information. In comparison to RS, DEAL reduces the labels required to reach peak accuracy from 150 to 60, resulting in a decrease of label complexity of $\approx 60\%$.
We also put emphasis on the quickness of DEAL in adapting to NNC early and to US

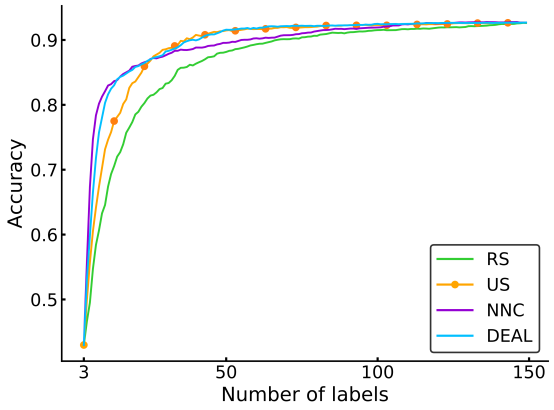


Figure 5.3: Comparing different query strategies on the real-world data of section 5.1 using a SVM with polynomial kernel of degree three. DEAL uses US and NNC as base query strategies. All three query strategies perform clearly better than sampling randomly (RS). Initial data is three samples and three classes. Averaged over 50 runs.

at the transition point.

Having chosen a classifier and proper query strategies we return to the original problem of finding the most efficient fraction value. We assume that

$$\text{number of failures per shift } n = 10 \tag{5.1}$$

and conclusively there are ten possible fraction values given by

$$\text{fraction value } p \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1\} \tag{5.2}$$

where, e.g., $p = 0.1$ corresponds to labeling one sample (out of the ten) per shift.
This allows us to track the classifier accuracy as a function of previous shifts to create figure 5.4.
Using the plots in figure 5.4 we can, once again, show the superiority of US over RS by computing the center of mass of the area under the different curves (AUC). This process assigns every curve two scalar values - the coordinates of the center of mass of the AUC. Given our 20 shifts of latter figure, it follows that the perfectly performing curve (a constant one function and its AUC a rectangle) has center of mass coordinates $(x, y) = (10.5, 0.5)$.
Plotting the x/y-coordinates of the center of masses versus fraction values p gives figure 5.5.
Since the best performing curve has the lowest possible x-value and highest possible y-value of the center of mass coordinates (COM), we can conclude that US performs better than RS. Of course we already knew that from figure 5.3. For now let's stick with only US for that reason and return to the actual problem of finding the ideal fraction value p.
In order to be able to gauge the accuracy gain of choosing a high fraction value against the associated additional labeling cost we introduce a cost function. This cost function gives us the expected expense for the next shift. It consists of the labeling costs and the cost induced by misclassification and we define it as

$$\mathcal{C}(c_1, c_2, p, n, a) \coloneqq c_1 pn + c_2(1 - a)n \tag{5.3}$$

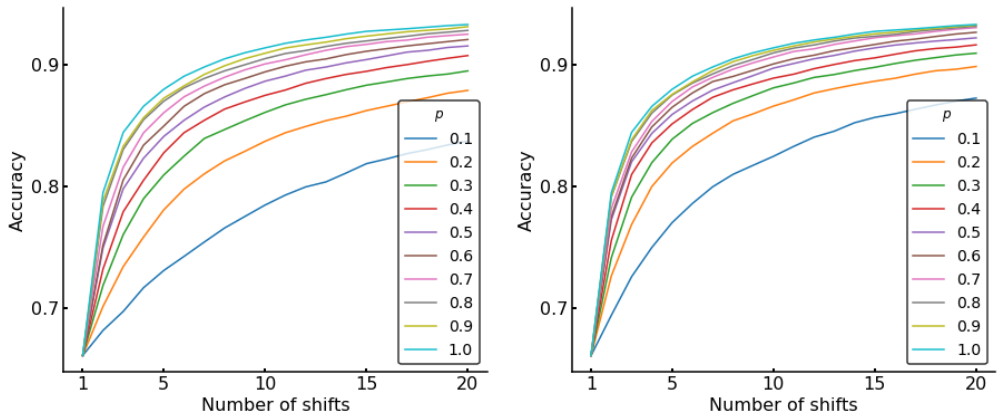


Figure 5.4.: Classifier accuracy as a function of previous shifts for different fraction values p. In the left figure the samples to label per shift are chosen randomly whereas in the right figure US proposes the samples to label. In the first shift all samples are labeled regardless of p to ensure large enough class diversity. Averaged over 500 runs.

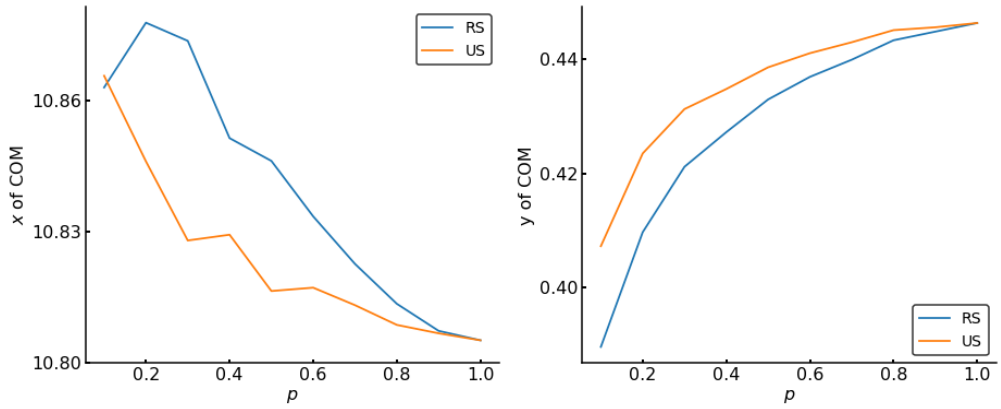


Figure 5.5.: Center of mass coordinates (COM) of the AUC of figure 5.4 versus the fraction values p. The y-axis of the left figure is the x-value of COM - the lower the better. The y-axis of the right figure is the y-value of COM - the higher the better. We can see US outperforming RS.

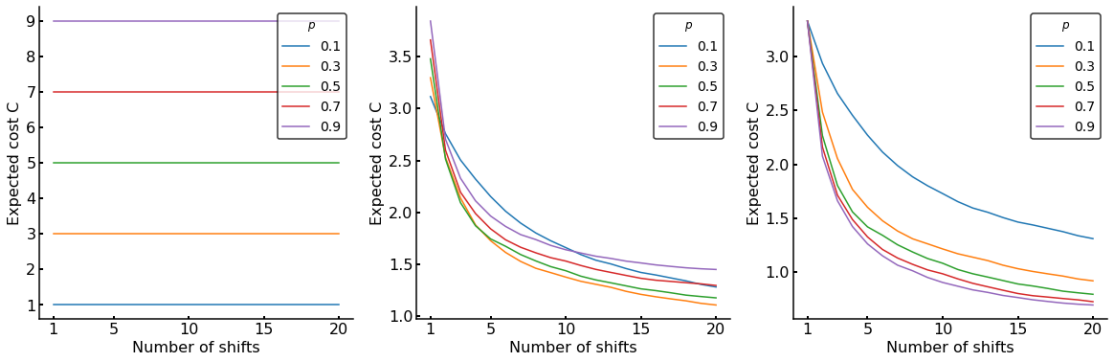


Figure 5.6.: The expected cost for the next shift as a function of the number of previous shifts for different fraction values p. The left, middle and right figure have cost ratio values of $c = 0, 0.9, 1$, respectively. Note that for $c = 0.9$ low- as well as high values of p give high expected cost.

where $c_1 \in \mathbb{R}$ is the cost for labeling one sample, $c_2 \in \mathbb{R}$ is the cost per misclassified sample/failure, p is the fraction value, n is the number of samples per shift and a is the accuracy of the classifier before the shift.
Substituting

$$c_1 = (1-c)b,\ c_2 = cb \quad \text{or} \quad b = c_1 + c_2,\ c = \frac{c_2}{b} \tag{5.4}$$

results in the convex combination

$$\mathcal{C}(c, b, p, n, a) = b\big((1-c)pn + c(1-a)n\big). \tag{5.5}$$

Since b is just an overall scaling factor it can be set to one without loss of generality. While this is also true for n it, in contrast to b, does play a role since a depends on n. This means that we end up with one cost ratio c where $c = 0$ corresponds to misclassification does not induce cost and $c = 1$ implies additional labels do not induce cost. To gain more intuition, take a look at figure 5.6. In the left figure c is zero thus misclassification does not induce cost and the expected cost is constant and only depends on the number of samples to label. In the right figure c is one and therefore additional labels are free so naturally high fraction values give low expected cost. Also the expected cost is decreasing since as the shifts progress the accuracy increases and consequently the number of misclassifications decreases.
The next logical step is to look at the accumulated cost over all shifts. This gives us for $c = 0.9$ the figure 5.7.
We can clearly see that the accumulated cost is minimal for $p = 0.3$, so what we have found is the optimal fraction value $p_{\min}$ given a cost ratio c.
Lastly, let's plot this optimal fraction value for different values of c. This gives figure 5.8. The right figure shows the saving in accumulated cost when using US instead of RS given a cost ratio. Especially in the realm of a miss classification inducing three times the cost of an additional label ($c = 0.75$) is the cost reduction most significant. In the

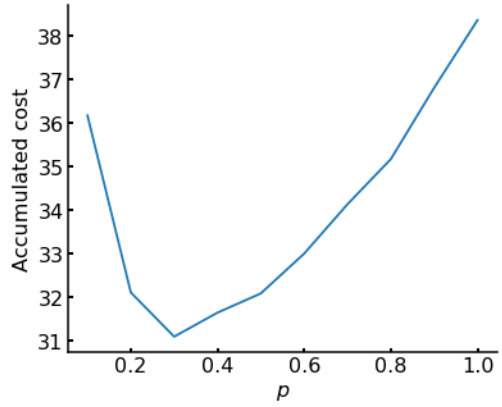


Figure 5.7: Plot of the accumulated cost over all twenty shifts as a function of the fraction value p for $c = 0.9$. We see that the accumulated cost is minimal for $p = 0.3$.

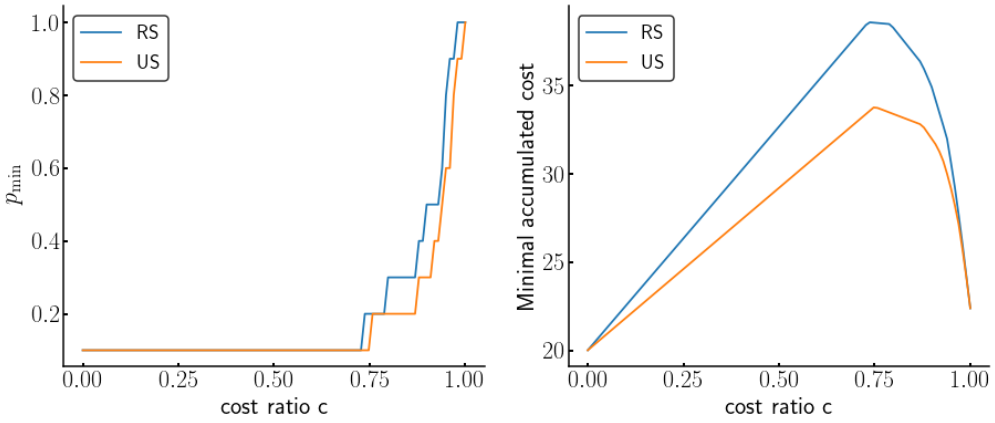


Figure 5.8.: Plot of the optimal fraction value and minimal accumulated cost against the cost ratio c. The minimal accumulated cost is the accumulated cost associated with $p_{\min}$. We, once again, see US outperforming RS.

left figure we also observe the optimal fraction value rapidly increasing as the cost ratio exceeds a threshold.

Overall, this allows us to gauge how many samples to label per shift, such that it minimizes the accumulated cost over 20 shifts given a cost ratio. Hence, the active learning framework used on the real-world data is completely determined with well performing query strategies (figure 5.3) and an approach to fine tune the fraction value.

5.3. Auxiliary pool

An alternative to active learning, with the goal of high label efficiency, is to generate an auxiliary pool using a (strongly) simplified labeling protocol. The hope is that after manipulating the auxiliary pool it is still capable of generalisation. This approach is beneficial if large quantities of labeled data are required.
In this section we use the techniques of chapter 3 to manipulate the auxiliary pool of our real-world data, such that it "solves" the classification problem that the primary pool poses.
In chapter 3 we discussed three possible approaches.
The first approach (section 3.4) is to allow the classifier to put less emphasis on some samples that are especially difficult to correctly predict. Thereby, we hope to eliminate the incorrect samples from the training pool.
The second approach (section 3.2) utilizes the much larger auxiliary pool to learn a sparse data encoding. By using the learned dictionary(encoding) to encode the primary pool we hope that the primary pool generalises better.
The third approach (section 3.5) tries to directly correct as many incorrect labels of the auxiliary pool as possible. Afterwards we treat the auxiliary pool as if it was perfectly labeled.

5.3.1. MLP with adapting sample weights

We allow the classifier to discard some samples by allowing an MLP to learn every sample's training weight. The training weight determines how much a sample's loss contributes. The details of the approach are outlined in section 3.4.
The MLP consists of one fully connected, hidden layer with ReLU activation function and an output layer with ten nodes and softmax activation. The general loss of a classifier with trainable weights is given in eq. (3.38) and it obviously depends on the vector of sample weights $\boldsymbol{w}$ and on the classifier's parameter vector $\boldsymbol{\theta}$. Since we use coordinate descent for optimization we are required to choose proper step sizes for optimization w.r.t. $\boldsymbol{\theta}$ and w.r.t. $\boldsymbol{w}$. We also have to choose the regularisation parameter $\lambda_1 > 0$ that punishes a lowering of a sample's training weight. The difficulty lies in choosing proper values for these hyperparameters.

For the results presented in table 5.1, the optimizer w.r.t. $\boldsymbol{\theta}$ or $\boldsymbol{w}$ is Adam, $\lambda_1 = 0.5$ and

[%]	Peak test accuracy at # of steps = 1500		# of steps = 5000	
constant weights	train accuracy 84.2 ± 0.19	test accuracy 90.4 ± 0.61	train accuracy 91.7 ± 0.048	test accuracy 76.1 ± 0.35
	Peak test accuracy at # of steps = 4000		# of steps = 6000	
adapting weights	train accuracy 80.9 ± 0.31	test accuracy 92.6 ± 0.26	train accuracy 82.4 ± 0.10	test accuracy 92.4 ± 0.29
	# weights < 0.1 20.7 ± 0.19	# weights > 0.9 78.2 ± 0.17	# weights < 0.1 19.8 ± 0.22	# weights > 0.9 79.6 ± 0.21

Table 5.1.: Training a MLP with poorly labeled, auxiliary pool and testing on the perfectly labeled, primary pool. We compare the accuracy gain when allowing the MLP to learn the sample weights. It is noteworthy that the MLP with adapting weights does not overfit.

step sizes are $\eta_{\boldsymbol{\theta}} = 10^{-4}, \eta_{\boldsymbol{w}} = 5 \cdot 10^{-4}$.
We find an increase in test accuracy of $\approx 2\%$ by allowing the classifier to learn sample weights.
It also discards (weight less than 0.1) about $\approx 20\%$ of the samples as indicated by the last row of the table. It is also interesting to note that almost no samples have weights between 0.1 and 0.9. This shows how well regularizing with an L1-norm works for learning a sparse training weight vector. This is crucial since samples are either incorrect and should not contribute or are correct and should contribute. There is no in-between.
Finally we note that, the MLP with adapting weights seems to overfit much less compared to the MLP with constant and equal sample weights. To see this, we establish that in the second column of table 5.1 the MLP with non-trainable weights reaches a test accuracy of $76.1 \pm 0.35\%$ despite a much higher training accuracy of $91.7 \pm 0.048\%$. In contrast to that, the MLP with trainable weights reaches a training accuracy of $82.4 \pm 0.10\%$ and a higher test accuracy of 92.4 ± 0.29.
This shows that by letting the MLP effectively neglect $\approx 20\%$ of the training data, it generalises better, meaning that it discards the samples with incorrect labels. We also note that the training accuracy is much lower, due to the fact that, again $\approx 20\%$ of the training data, doesn't actually participate in the training.

5.3.2. Dictionary learning

We deploy the auxiliary pool for learning a sparse representation of the real-world data. Then, we use the learned dictionary to transform the primary pool into a sparse format. By using the much larger auxiliary pool for learning a suitable encoding we hope to improve the primary pool's ability to generalise. The details of this approach are described in section 3.2.

We differentiate between three scenarios. In the first, we neither use the auxiliary pool

nor dictionary learning. In the second, we use dictionary learning but not the auxiliary pool. In the third, we use the auxiliary pool for learning the dictionary and afterwards transform the primary pool with the dictionary.
To compare in which scenario the primary pool generalises best, we estimate the test accuracy of five different classifiers. We estimate the test accuracy by splitting the primary pool into a 200 samples large training pool and use the remaining 142 samples as test data. We repeat this procedure 20 times and estimate the average and standard deviation.
This produces the results given in table 5.2 with parameters given in the corresponding parameter sheet. For the L1-regularization parameter we choose $\lambda = 1$. This parameter determines how strongly the data is forced into a sparse representation.

Examining the results of table 5.2 we find a slight improvement in the test accuracies in scenario c) compared to b). This means, that using the auxiliary pool for learning the dictionary is beneficial to learning the dictionary from the primary pool. However, both approaches are far inferior to scenario a), where we don't use dictionary learning and leave the primary pool in its original representation. Dictionary learning does not seem to work well on this dataset.

[%]	a) no DL	b) DL **without** auxiliary pool	c) DL **with** auxiliary pool
linear	93.6 ± 1.62	87.1 ± 2.85	87.5 ± 1.81
polynomial	93.2 ± 1.61	79.7 ± 2.84	81.5 ± 3.46
rbf	91.5 ± 1.58	84.4 ± 1.61	86.0 ± 1.67
RFC	94.5 ± 1.37	84.2 ± 2.46	84.5 ± 3.21
MLP	94.2 ± 1.22	88.0 ± 2.79	88.1 ± 2.64

Table 5.2.: Test accuracy of five classifiers on the primary pool which is split into 200 training and 142 test samples. In each column the primary pool is in a different representation, in a) is the unmodified primary pool, in b) the transformed primary pool using a dictionary learned by the primary pool and in c) the transformed primary pool using a dictionary learned by the auxiliary pool. Given is the mean of 20 runs, ± one standard deviation.

Parameter sheet for table 5.2

Datasets: Primary pool (1), auxiliary pool (2)

Dataset Number	1	2
Number of samples	342	6703
Number of features	36	36
Number of classes	9	10[a]
Number of training samples	200	-
Number of test samples	142	-
Number of runs for averaging	20	-

Classifier:

- SVM with linear kernel function
- SVM with polynomial kernel function
- SVM with rb kernel function
- RFC with 100 estimators
- MLP with one (fully connected) hidden layer with 100 nodes and ReLU activation, output layer with 10 nodes and softmax activation

[a] This is not a typo. Due to the small size of the primary pool, there is one (less frequent) class not present at all.

5.3.3. Label correction

In this section we apply the four label correction techniques from section 3.5 to correct the labels from the auxiliary pool. Afterwards we treat the auxiliary pool as if it were perfectly labeled. We estimate the test accuracy of different classifiers trained on the auxiliary pool before and after correction. We use the primary pool, which is perfectly labeled, to estimate test accuracies. We expect to see an improvement in the test accuracies if the label correction algorithms successfully correct enough labels.
By using the primary pool for estimating the test accuracy, we estimate the performance of the different classifiers on the actual classification problem.
The four label correction algorithms are NNC, ADE, CC and BCC. The parameters of the label correction methods are as follows:

- NNC: $K = 8, p = 0.5$
- ADE: $N_{\text{epoch}} = 50, N_{\text{era}} = 150, \eta_{\boldsymbol{\omega}} = 10^{-5}, \eta_{\boldsymbol{p}} = 10^{-3}, \epsilon = 10^{-5}$
- CC: $A = 25$
- BCC: –

The before- and after-label correction test- and train accuracies are given in table 5.3 for the different label correction methods and five different classifiers. The parameters of the classifiers are given in the parameter sheet for table 5.2.

Let's go through the results of the different label correction algorithms one by one.
The novel NNC performs very well. NNC substantially increases the test accuracy of all five classifiers. When solely relying on the the auxiliary pool as training data we are able to reach a test accuracy of almost 95% by using NNC. As an example, when we choose a linear SVM as classifier we see an increase in test accuracy of above 16%, in case of a RFC we find an increase in test accuracy of almost 20%. This is especially impressive when we consider the simplicity of NNC and its robust hyperparameters as a consequence thereof. We recall that NNC managed to deliver impressive results on the toy datasets (section 4.4.1) with the same hyperparameters as chosen here.
ADE delivers, after NNC, the second best performance. It also manages to increase the test accuracy of all five classifiers significantly. These positive results strengthen the believe that ADE is a powerful but non-robust label correction algorithm that suffers from hyperparameter optimisation. We already noted that, when applying ADE to toy datasets (section 4.4.2), it struggled for two out of the three datasets for which its hyperparameter were not fine-tuned. However, in this section and with properly chosen hyperparameter, ADE, yet again, manages to produce good results.
CC is the weakest performer of the four label correction algorithms. While it still delivers noticeable test accuracy increases for four out of the five classifiers, it is the only label correction algorithm that leads to a decrease in test accuracy. In case of the MLP test accuracy decreases. Overall, these below average results, when compared to the other label correction algorithms, is contrary to the results of section 4.4.3 where CC has proven to be the second best algorithm.

BCC manages to increase test accuracy for all classifiers, even though the increase is negligible in case of the MLP. Albeit, an increase of, e.g., ≈ 9% in case of the RFC is impressive, considering BCC does not require any hyperparameters. This makes it truly an out-of-the-box algorithm.

As a remark, applying NNC to the auxiliary pool leads to a label change of 16.7% of all samples in the auxiliary pool. The auxiliary pool with the label configuration induced by NNC delivers the highest test accuracy on the primary pool (throughout all classifiers!). As a consequence, it is the label configuration that is the most aligned with the primary pool, with perfectly labeled data. This label configuration is the closest we get to "perfectly labeled". Therefore, our best estimate for the number of incorrect labels in the original auxiliary pool is equal to the number of changed labels when applying NNC, which was 16.7% of all samples in the auxiliary pool.
In figure 5.9 we take a closer look at this label configuration, the label configuration that results from label correction using NNC.
The upper two plots display the confusion matrix of a linear SVM on the primary pool. We can nicely see the increase in test accuracy by using NNC. We completely eliminate the mismatches where the true label is "machine five" or "machine four". In the lower two plots we can see how NNC achieves this - by introducing homogeneity in certain regions in data space. E.g., before label correction the bottommost cluster is cluttered with additional labels "machine 4", "machine 3", "machine 2" and "machine 1". After label correction this cluster is more homogeneous and the mismatches for samples with true label "machine 5" disappear in the confusion matrix b).

Overall, it is impressive how well label correction and especially NNC perform. After all, a linear SVM trained on the auxiliary pool with corrected labels using NNC is able to reach a test accuracy of almost 95% on the primary pool. Even though the labels of the auxiliary pool are obtained without human effort whereas the labels of the primary pool are labeled by a human.

[%] ↘	a) original labels train accuracy	a) original labels test accuracy	b) corrected labels train accuracy	b) corrected labels test accuracy
		NNC		
linear	90.5	78.6	95.7	94.7
poly	91.7	78.1	97.6	92.7
rbf	90.1	81.9	96.5	94.4
RFC	100.0	73.1 ± 0.48	100.0	93.0 ± 0.32
MLP	86.2 ± 0.19	88.3 ± 0.38	93.2 ± 0.13	93.8 ± 0.19
		ADE		
linear	90.5	78.6	98.4 ± 0.38	87.7 ± 1.73
poly	91.7	78.1	98.6 ± 0.29	87.8 ± 1.79
rbf	90.1	81.9	97.2 ± 0.61	90.1 ± 1.49
RFC	100.0	73.1 ± 0.48	100.0	88.2 ± 1.93
MLP	86.2 ± 0.19	88.3 ± 0.38	94.6 ± 1.07	92.4 ± 0.94
		CC		
linear	90.5	78.6	94.8 ± 0.47	84.0 ± 1.49
poly	91.7	78.1	97.8 ± 0.26	83.2 ± 1.35
rbf	90.1	81.9	97.2 ± 0.27	83.5 ± 1.37
RFC	100.0	73.1 ± 0.48	100.0	83.4 ± 1.72
MLP	86.2 ± 0.19	88.3 ± 0.38	93.2 ± 0.58	83.6 ± 1.69
		BCC		
linear	90.5	78.6	89.7 ± 0.36	83.9 ± 0.82
poly	91.7	78.1	93.4 ± 0.30	83.5 ± 0.83
rbf	90.1	81.9	91.4 ± 0.24	87.1 ± 0.80
RFC	100.0	73.1 ± 0.48	100.0	82.1 ± 1.52
MLP	86.2 ± 0.19	88.3 ± 0.38	86.5 ± 0.36	88.4 ± 0.74

Table 5.3.: Accuracy of different classifiers before- and after label correction. Column a) corresponds to classifiers trained on the unchanged auxiliary pool and column b) on the auxiliary pool after label correction using NNC, ADE, CC and BCC. The primary pool is used as testing data. The parameters of the classifiers are (again) given in parameter sheet for table 5.2. Empirical standard deviations that are not given are $\leq 10^{-10}$. Given is the mean of 20 runs, ± one standard deviation.

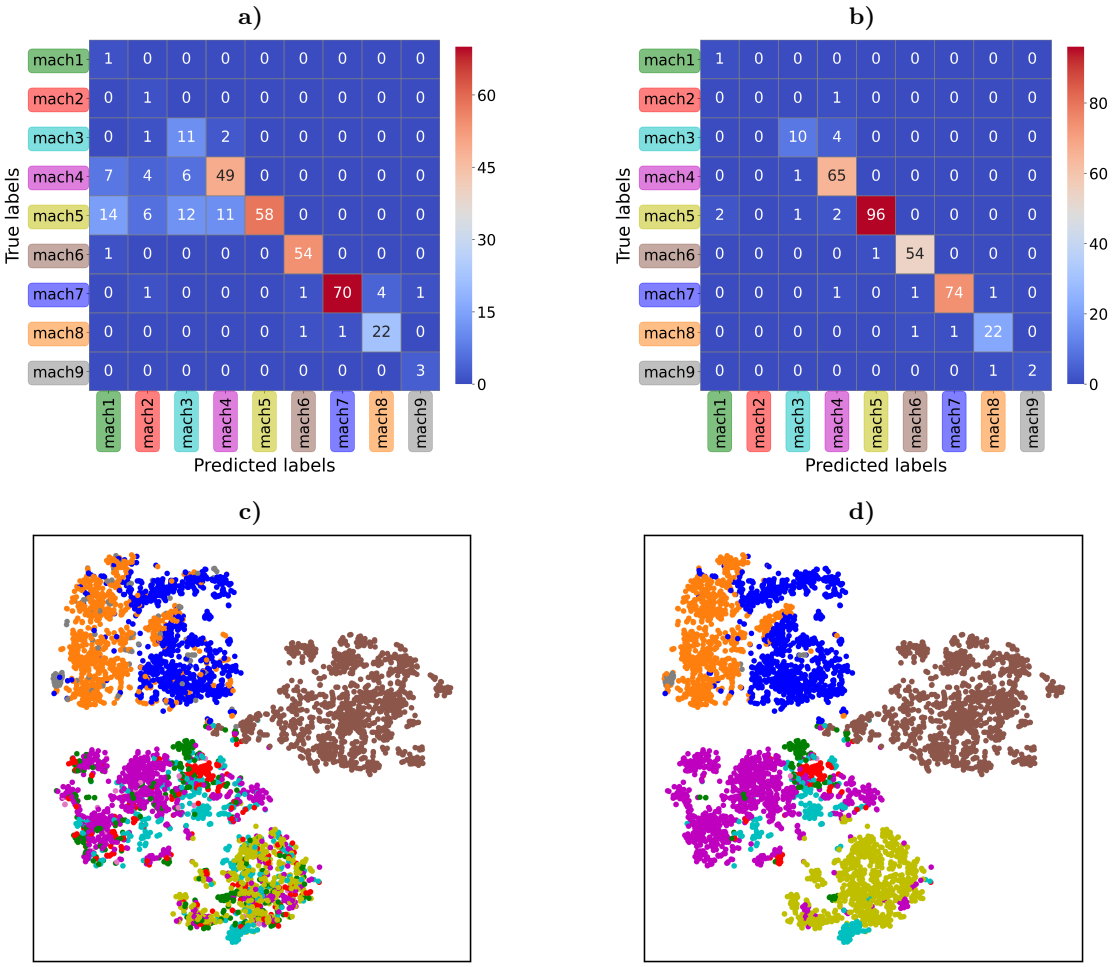


Figure 5.9.: Confusion matrix of label predictions on test data and labels of auxiliary pool before and after label correction using algorithm 20. Subfigure a) shows the confusion matrix of the label predictions on the primary pool where the label predictions are obtained using a linear SVM trained on the auxiliary pool with original labels, similarly subfigure b) uses the same classifier trained on the auxiliary pool with corrected labels. Plot c) and d) is a 2D-representation of the auxiliary pool generated using TSNE with original and corrected labels, respectively.

5.4. Guideline

In this section we talk about reference points on, if and how to use active learning on other datasets and whether or not to exploit an auxiliary data source.

5.4.1. Active learning

Q: When should you consider using active learning?
A: You should consider active learning whenever it is not possible/desirable to label all data. This might be due to the sheer amount of unlabeled data (big data) or due to a lack of resources (not enough man power, limited evaluation facilities, ..). Bottom line: Whenever labeling is expensive and one can afford not using each sample. The smaller the ratio of samples you are able to label, the more beneficial active learning can be.

Q: What are the up- and downsides of active learning?
A: The upside is that, if you choose the subset to labels wisely (by using active learning), you can be rewarded with a much better performing classifier, than if you choose the subset randomly. The major downside of active learning is the risk of ending up with a subset that suffers severely from sampling bias. Meaning that it will not generalise well as it is not representative of all data.

Q: Pool-based or on-line active learning?
A: On-line active learning certainly has advantages - there is no unlabeled pool, exploration can be built-in using coin toss and it may be easier to implement in certain environments. Still, pool-based active learning seems more robust with larger label complexity decreases overall. At the end, this is highly dependent on the problem at hand.

Q: Which query strategies to use?
A: It is beneficial to combine several query strategies together using ALBL or DEAL. Uncertainty sampling should always be one of them, as it performs well later on. Good explorative options are RS, NNC or ReS (space = 'labeled', goal = 'low'). You may want to add CBS for a balanced training dataset.

Q: My data is very high-dimensional. Will this lead to problems with, e.g., NNC?
A: Yes and no. If your data is very high-dimensional the concept of nearest neigh-

bours will lose meaning (curse of dimensionality) but you are free to use an alternative representation with reduced dimensions for NNC while using the full representation to train the classifier. NNC does not depend on the classifier. The same argument holds true for ReS and MdS.

5.4.2. Auxiliary data source

Q: When should you consider utilizing an auxiliary data source?
A: If you know beforehand that training your classifier to a sufficient degree will require lots of labeled data despite using active learning then an auxiliary data source is appropriate. The reason is, that we assume that a first (imperfect) label estimate is cheap. This makes the requirement of many labeled samples less cumbersome. Of course, this will lead to a subpar label accuracy which may not be a problem though. As we may be able to correct (most) of the incorrect labels using label correction methods.

Q: Are you able to generate labels cheaply?
A: To create an auxiliary pool you need a cheap way of labeling data. Cheap enough, so that you are able to label enough data while still maintaining a correct label rate of at least 50% (better 60%). This may be, e.g., replacing the human by an artificial entity or simplifying the labeling-routine in some crucial steps.

Q: Can you modify your classifier to support learn-able weights?
A: A classifier should be able to learn sample weights, if the loss it optimizes, is using M-estimation and the learning procedure uses some form of gradient descent (or 2nd order techniques).
As a side note: The weight vector should be L1-Regularized, since ideally all incorrectly labeled samples are discarded while the weight of all correctly labeled samples is unchanged. Hence, a sparse weight vector is favorable.

Q: Will label correction methods work on my data?
A: All methods seem to perform best when the data to correct has a false label rate between 0.2 and 0.4. Of course this quantity may be unknown and quite hard to gauge.

Q: Which label correction to choose?
A: NNC (almost) always performed very well and requires only few, quite intuitive

hyperparameters that need to be set. CC and BCC also only require one or no hyperparameter, respectively, making them easy to use. Unfortunately they both perform poorly in the regime of low false label rates ($\beta \leq 0.1$) which may not be excludable beforehand. ADE's performance strongly depends on well chosen hyperparameters making it difficult to use whilst returning only average label correction performance.

Q: What if my data is high-dimensional?
A: If your data is very high-dimensional the approaches NNC, CC and BCC, as they are defined, may not work well due to the fact that they all rely on Euclidean distance as a similarity measure. Therefore, it is then necessary to replace the similarity measure by one that has proven meaningful on your data despite its high-dimensional nature. Alternatively reducing dimensions beforehand (only for the label correction/pre-processing part) may be a valid option.

6. Conclusion

In this final chapter we will summarize the findings and access the overall usefulness of the involved algorithms. Before discussing the results, we quickly recap.

The goal of this thesis can be outlined very easily: We want to decrease the cost associated with labeling data, labeled data that is required for supervised classification.
There are two major approaches for cost-cutting. We either try to label less or label more cheaply. The task of reaching similar classifier accuracy with less labeled data is tackled by active learning. In the second approach, we decrease labeling cost by labeling more cheaply using an auxiliary data source. This comes at the cost of an inferior label accuracy. We discuss methods of manipulating this flawed data such that it becomes useful again. We put emphasis on label correction where we try to correct as many false labels as possible.

We start of discussing the results by focusing on active learning, specifically pool-based active learning. In pool-based active learning a, so called, query strategy proposes the sample to label next out of the pool of unlabeled data that is currently available. Based on the criteria a query strategy uses to choose a certain sample, query strategies can be allocated into three categories, into representative-, informative- or performance-based query strategies. When applying pool-based active learning to artificial- and toy datasets, it manages to decrease label complexity significantly, e.g., in case of the Wine dataset by $\approx 65\%$. Label complexity counts the number of labels required to reach a certain accuracy. I.e., in case of the Wine dataset, to reach the same accuracy of a classifier trained on 35 samples chosen by pool-based active learning, one needs to label at least 100 randomly chosen samples. Pool-based active learning delivers equally impressive results when applied to real-world data. It manages to reduce label complexity by up to 60%, more than halving the labeling efforts. Further, we find that even though numerous query strategies are introduced, only few query strategies are sufficient to continuously outperform all remaining ones. These query strategies are US as the informative-based- and NNC as the representative-based query strategy. As as result, the top of the line approach to pool-based active learning is using DEAL to dynamically combine RS, US and NNC. RS is added as a fallback query strategy, ensuring that the combined approach performs as least as good as choosing samples randomly.
As part of the work a python-package titled "poolAL", that implements all discussed query strategies in an easy-to-use fashion, was created and made accessible for everyone.

Finally, we discuss the results of utilizing an auxiliary data source that frequently mislabels samples. We focus on using, in total, four different label correction algorithms to

correct flawed samples. When applied to artificially falsified toy datasets, the algorithms NNC and CC manage to deliver impressive results. E.g., in case of the EMNIST-Digits dataset both are able to almost completely invert the falsification process and retrieve the original labels, even when half the data is incorrect. Further, we find that label correction algorithms can lead to a higher classifier test accuracy after label correction even when no data is previously falsified. When applied to real-world data, all four label correction algorithms manage to increase the test accuracy of different classifiers that are trained on the auxiliary data. The novel NNC outperforms all other label correction algorithms for every classifier significantly. E.g., training a RFC on the auxiliary data leads to a test accuracy of $73.1 \pm 0.48\%$. Using NNC on the auxiliary data and afterwards training the same RFC on the auxiliary data reveals a test accuracy of $93.0 \pm 0.32\%$, an increase of almost 20%. Additionally, by using NNC a linear SVM is able to reach a test accuracy of almost 95%. A test accuracy that is estimated on data that was labeled by a human, whereas the labeling of the auxiliary pool was not performed by a human and obviously NNC does not involve human effort. Again, as part of the work, a python-package that implements the different label correction algorithms was created and made accessible for everyone.

Overall, both active learning and an auxiliary data source can prove to be an effective way of reducing labeling efforts. Considering that this is relevant to any machine learning practitioner unable to label all available data, the topic concerns many fields, especially in real-world application.

In the future both sections can be extended by additional query strategies or label correction algorithms as there is still a cornucopia of ideas available in often rather unknown papers. Additional testing and application to various data is necessary to better understand the different query strategies/label correction methods and their respective fields of application.

List of Figures

List of Tables

Bibliography

[1] Charu C. Aggarwal, Xiangnan Kong, Quanquan Gu, Jiawei Han, and Philip S. Yu. *Data Classification: Algorithms and Applications*. CRC Press, London, New York, 2014.

[2] M. F. Balcan, A. Z. Broder, and T. Zhang. Margin based active learning. In *In Proceedings of the 20th Annual Conference on Learning Theory*, page 35–50, 2007.

[3] Yoram Baram, Ran El-Yaniv, and Kobi Luz. Online choice of active learning algorithms. *Journal of Machine Learning Research 5*, 5:255–291, 03 2004.

[4] Alina Beygelzimer, Sanjoy Dasgupta, and John Langford. Importance weighted active learning. *CoRR*, abs/0812.4952, 2008.

[5] Alina Beygelzimer, John Langford, Lihong Li, Lev Reyzin, and Robert E. Schapire. Contextual bandit algorithms with supervised learning guarantees, 10 2010.

[6] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters, 2017.

[7] I. Dagan and S. P. Engelson. Committee-based sampling for training probabilistic classifiers. In *In Proceedings of the 12th International Conference on Machine Learning*, pages 150–157, 1995.

[8] S. Dasgupta and D. Hsu. Hierarchical sampling for active learning. In *In Proceedings of the 25th International Conference on Machine Learning*, page 208–215, 2008.

[9] Sanjoy Dasgupta. Coarse sample complexity bounds for active learning. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 235–242. MIT Press, San Diego, 2006.

[10] P. Donmez, J. G. Carbonell, and P. N. Bennett. Dual strategy active learning. In *In Proceedings of the 18th European Conference on Machine Learning*, page 116–127, 2007.

[11] B. Du, Z. Wang, L. Zhang, L. Zhang, W. Liu, J. Shen, and D. Tao. Exploring representativeness and informativeness for active learning. *IEEE Transactions on Cybernetics*, 47(1):14–26, 2017.

[12] R.A. Fisher and Michael Marshall. Wine dataset. `https://scikit-learn.org/stable/datasets/index.html#wine-dataset`, 2013. Last accessed: 2020-01.

[13] Y. Freund, H. S. Seung, E. Shamir, and N. Tishby. Selective sampling using the query by committee algorithm. *Machine Learning Volume 28, 2-3*, pages 133–168, 1997.

[14] Wei-Ning Hsu and Hsuan-Tien Lin. Active learning by learning. In *In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, page 2659–2665, 01 2015.

[15] Sheng jun Huang, Rong Jin, and Zhi-Hua Zhou. Active learning by querying informative and representative examples. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 892–900. Curran Associates, Inc., 2010.

[16] D. D. Lewis and J. Catlett. Heterogeneous uncertainty sampling for supervised learning. In *In Proceedings of the 11th International Conference on Machine Learning*, page 148–156, 1994.

[17] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *In Proceedings of the 17th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12, 1994.

[18] Xuejun Liao, Ya Xue, and Lawrence Carin. Logistic regression with an auxiliary data source, 01 2005.

[19] Edwin Lughofer. On-line active learning: A new paradigm to improve practical useability of data stream modeling methods. *Information Sciences*, 415, 06 2017.

[20] Kevin P. Murphy. *Machine Learning - A Probabilistic Perspective*. The MIT Press, Cambridge, Massachusetts and London, England, 2012.

[21] H. T. Nguyen and A. W. M. Smeulders. Active learning using pre-clustering. In *In Proceedings of the 21st International Conference on Machine Learning*, page 623–630, 2004.

[22] Bryce Nicholson, Victor Sheng, Jing Zhang, and Zhiheng Wang. Label noise correction methods, 10 2015.

[23] Kunkun Pang, Mingzhi Dong, Yang Wu, and Timothy M. Hospedales. Dynamic ensemble active learning: A non-stationary bandit with expert advice, 2018.

[24] M. Prasad and Arcot Sowmya. Multi-class unsupervised classification with label correction of hrct lung images, 02 2004.

[25] Quartl. P-norm. `https://de.wikipedia.org/wiki/P-Norm`, 2011. Last accessed: 2020-01.

[26] Daniel Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, and Zheng Wen. A tutorial on thompson sampling, 2017.

[27] Burr Settles. Active learning literature survey, 01 2010.

[28] H. S. Seung, M. Opper, and H. Sompolinsky. Query by committee. In *COLT '92: Proceedings of the fifth annual workshop on Computational learning theory*, pages 287–294, July 1992.

[29] Tensorflow. Tensorflow playground. `http://tensorflow.playground.org`, 2020. Last accessed: 2020-01.

[30] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. In *In Proceedings of the 17th International Conference on Machine Learning*, page 999–1006, 2000.

[31] Sriram Vajapeyam. Understanding shannon's entropy metric for information, 2014.

[32] Zheng Wang and Jieping Ye. Querying discriminative and representative samples for batch mode active learning. *ACM Transactions on Knowledge Discovery from Data*, 9:158–166, 08 2013.

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

[34] Z. Xu, K. Yu, V. Tresp, X. Xu, and J. Wang. Representative sampling for text classification using support vector machines. In *In Proceedings of the 25th European Conference on Information Retrieval Research*, page 393–407, 2003.

[35] Yazhou Yang. Towards practical active learning for classification, 2018.

[36] Xinchuan Zeng and Tony Martinez. An algorithm for correcting mislabeled data. *Intell. Data Anal.*, 5:491–502, 12 2001.

A. Introduction into supervised-ML

Typically, a computer determines an output based on the input *and* a set of rules. Depending on the connection between input and output, finding a satisfactory set of rules can be problematic.
ML is concerned with finding such a set of rules for you.
Supervised-ML determines a set of rules based on corresponding pairs of input and output. Therefore, we have to provide a sufficient number of such pairs. This data is called training data.
After supervised-ML has established a mapping between the input-output pairs from the training data, we can use this mapping to determine the output based on an arbitrary input. In the spirit of "a picture is worth a thousand words", we summarize supervised-ML in figure A.1.

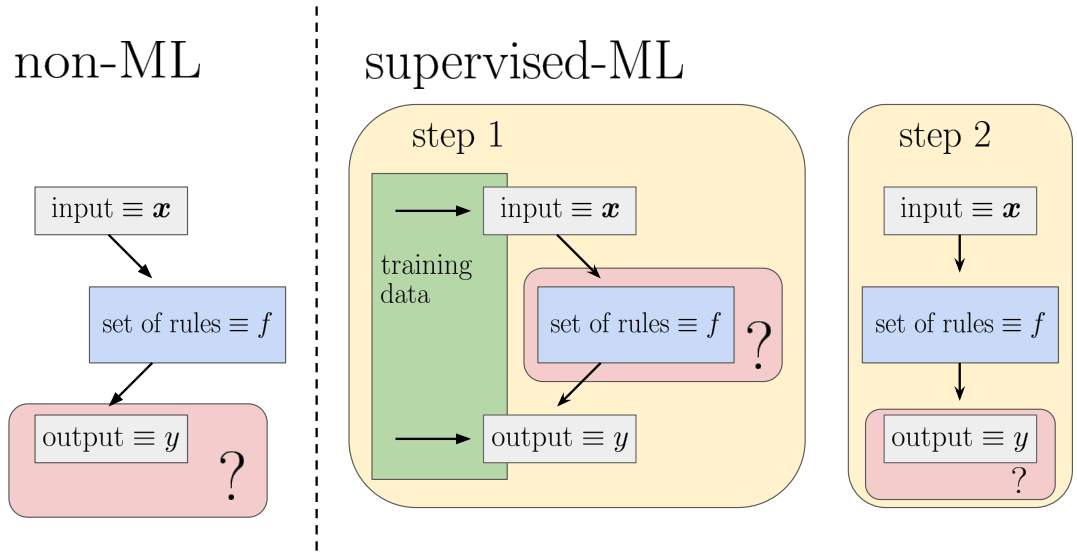


Figure A.1.: Summarizing supervised-ML. Supervised-ML uses the training data to learn a mapping between input and output. Then, the learned mapping can be used to compute the output of an arbitrary input.

Q: How to find a good set of rules given the training data?
A: In short: Loss optimisation
We assume that $\mathcal{F}$ is the model with parameters $\boldsymbol{\theta}$ that maps input to output. Note that choosing an appropriate model is non-trivial. Then, we can use the model to make a prediction

$$y_{\text{prediction}} = \mathcal{F}(\boldsymbol{x}, \boldsymbol{\theta}). \tag{A.1}$$

In the case of training data the "true" output is known. Thus, we choose a measure to quantify the mismatch between prediction and truth - the lower the better. We call this measure loss and denote it by

$$\mathcal{L}(y_{\text{prediction}}, y_{\text{true}} \equiv y) \equiv \mathcal{L}(\boldsymbol{x}, y, \boldsymbol{\theta}). \tag{A.2}$$

Ideally, we are interested in the *expected* loss, so

$$\overline{L}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x},y}[\mathcal{L}(\boldsymbol{x}, y, \boldsymbol{\theta})] \tag{A.3}$$

which we can estimate using Monte-Carlo integration, resulting in

$$\hat{\overline{L}}(\boldsymbol{\theta}) \propto \sum_{\{\boldsymbol{x},y\} \in \text{ training data}} \mathcal{L}(\boldsymbol{x}, y, \boldsymbol{\theta}). \tag{A.4}$$

The loss constrained to the training data is a functional assigning every point in parameter space a scalar value. By minimizing the constrained loss, we find good parameters for our model, i.e.,

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \hat{\overline{L}}(\boldsymbol{\theta}). \tag{A.5}$$

Finally, as an example for loss optimisation let's take a look at linear regression. The example then concludes this brief introduction.

A.1. Example: Linear regression

Task: Fit a line through one-dimensional data points

Parametric model: $y_{\text{prediction}} \coloneqq \mathcal{F}\big(x, \boldsymbol{\theta} = (m, b)^T\big) = mx + b$

Squared-error loss: $\mathcal{L}(y_{\text{prediction}}, y) \coloneqq (y_{\text{prediction}} - y)^2$

Mean-squared-error loss: $\hat{\overline{L}}(m, b) \coloneqq \frac{1}{L} \sum_{\{\boldsymbol{x},y\} \in \text{ training data}} ((mx + b) - y)^2$ where L is number of training pairs

As an example we randomly draw 50 samples from $\propto \mathcal{U}\mathit{niform}([0, 5])$, we then let $y = 0.5x$ plus some random Gaussian noise.
Figure A.2 displays the MSE constrained to the 50 samples in parameter space. Furthermore, two explicit models corresponding to two points in parameter space are plotted in figure A.3.

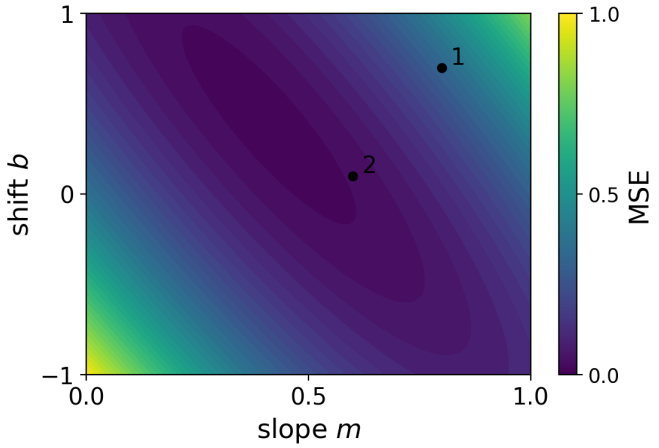


Figure A.2.: MSE assigns every point in parameter space a scalar value. Note that it is small in the region around $m = 0.5, b = 0$.

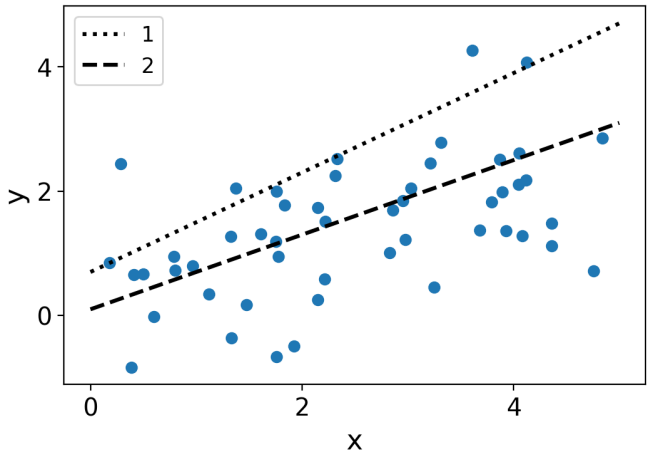


Figure A.3.: Linear regression on toy example. The two lines correspond to the two points in parameter space in figure A.2. Line 2 results in a smaller MSE and also visually fits the data better.

An dieser Stelle möchte ich mich gerne bei all denjenigen bedanken, die mich bei dem Erstellen dieser Masterarbeit unterstützt haben.

Zuerst gilt mein Dank Prof. Dr. Elmar Lang, der mir diese Masterarbeit ermöglicht und betreut hat.

Ebenfalls gebührt mein Dank Herrn Marinus Bommer, der mich geduldig unterstützt hat und dessen Daten ich nutzen durfte.

Außerdem bedanke ich mich bei meinen Büro-Kollegen für die herzliche Aufnahme und gute Atmosphäre.

Vielen Dank an Herrn Jonas Paletschek für ausgiebige Gespräche und anschließendes Korrektur lesen.

Abschließend möchte ich mich bei meinen Eltern bedanken, die mir mein Studium durch ihre Unterstützung ermöglicht haben.

Simon Bachhuber,
Regensburg

Ich habe die Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und bisher keiner anderen Prüfungsbehörde vorgelegt. Außerdem bestätige ich hiermit, dass die vorgelegten Druckexemplare und die vorgelegte elektronische Version der Arbeit identisch sind, dass ich über wissenschaftlich korrektes Arbeiten und Zitieren aufgeklärt wurde und dass ich von den in §24 Abs. 5 vorgesehenen Rechtsfolgen Kenntnis habe.

Regensburg