University of Magdeburg

School of Computer Science

Bachelor Thesis

Estimating hold-out-sample Performance for Active Learning

Author:

Florian Bethe

December 23, 2015

Reviewer:

Prof. Myra Spiliopoulou

Department of Technical and Operative Information Systems

Supervisors:

Daniel Kottke

Dr. Georg Krempl

Bethe, Florian:
Estimating hold-out-sample Performance for Active Learning
Bachelor Thesis, University of Magdeburg, 2015.

Abstract

In dieser Arbeit wird die Fragestellung beleuchtet, ob eine Kombination von Kurvenregression und Kreuzvalidierung die Leistung eines Klassifikators bei Benutzung verschiedener aktiver Lerner ohne systematische Abweichungen schätzen kann, besonders wenn noch nicht viele Trainingsinstanzen vorliegen. Zu diesem Zwecke untersuche ich vier verschiedene Verfahren auf Basis von *leave-one-out Kreuzvalidierung* sowie verschiedenen Funktionsmodellen vor. Mit in Betracht gezogen werden auch die Streuung der Schätzungen sowie die benötigte Rechenzeit. In simulierten Tests werden die Kombinationen für verschiedene Datensätze mit bereits erprobten Schätzern verglichen.

In this work we evaluate the question if a combination of curve fitting and cross-validation produces an estimator without systematic error capable of assessing a classifier's performance, especially with few purchased training instances. For this purpose, I examine four methods based on *leave-one-out cross-validation* as well as different function models. Also of interest are the spread of the error made as well as the necessary computation time. The methods are tested and compared against state-of-the-art estimators.

Acknowledgements

First, I would like to thank my supervisors, Dr. Georg Krempl and Ms. Sc. Daniel Kottke, for their continuous support and expenditure of time throughout the process of writing this thesis, without which I could not have finished it.

I would also like to thank both Dennis Kammerhoff and Sven Kalle for enduring my complaints and their moral support.

Another ”thank you” goes to my family, especially my sister Alexandra, also for their help and support.

Last but not least, I want to thank the staff of the assay office for dispelling my questions and uncertainties of just about every aspect of my studies.

Contents

Acknowledgements v

List of Figures x

List of Tables xi

List of Algorithms xiii

1 Introduction 1

2 Background and Related Work 3
- 2.1 Classification 3
- 2.2 Active learning 4
  - 2.2.1 Random sampling 4
  - 2.2.2 Uncertainty sampling 5
  - 2.2.3 Probabilistic Active Learning (PAL) 5
  - 2.2.4 Model Retraining Improvement and Monte-Carlo-Sampling . . . 6
- 2.3 Generalization Performance 7
  - 2.3.1 (Expected) prediction error, training error and loss 7
  - 2.3.2 Bias and variance 8
  - 2.3.3 Classifier-based estimators 9
  - 2.3.4 Cross-Validation 10
  - 2.3.5 Bootstrapping 11
- 2.4 Learning Curves and Regression Models 13
  - 2.4.1 Function families 13
  - 2.4.2 Maximum-likelihood regression 14

3 Modeling performance estimation as a learning process 15
- 3.1 Performance estimation on training sub-sets 15
  - 3.1.1 Sub-sampling of fitting points 17
  - 3.1.2 Extracting learning curves from performance estimates 19
- 3.2 Combining sub-estimates with curve fitting 20
  - 3.2.1 Function models and fitting algorithms 20
  - 3.2.2 Tuning the model fitting 21

4 Evaluation **25**
4.1 Performance criteria . . . 25
4.2 Method selection . . . 27
4.3 Test environment . . . 28
4.3.1 Reference accuracy . . . 28
4.3.2 Function fitting . . . 28
4.3.3 Active learner . . . 28
4.3.4 Classifier . . . 29
4.3.5 Datasets . . . 29
4.3.6 Parameters . . . 31
4.4 Test results . . . 32
4.4.1 Average Mean Error . . . 33
4.4.2 Average squared error . . . 36
4.4.3 Kullback-Leibler divergence . . . 38
4.4.4 Computation Time . . . 39

5 Conclusion and Future Work **41**

Bibliography **43**

List of Figures

2.1 PAL illustration showing the weighted probabilistic gain (normed on $[0, 1]$). The pgain is highest in areas where no labels have been purchased yet . . . 6

2.2 Classifier state for random and uncertainty sampling as well as PAL for 16 purchased labels on checke1 . . . 7

2.3 General appearance of a learning curve. Training set size as x-axis, accuracy as y-axis [17] . . . 14

3.1 Left: Three example paths generated with path-superset sub-sampling. The zigzag shape illustrates the difficulties for the fitting. Right: All possible accuracies for training subsets with $k = 6$ . . . 18

3.2 The shape of the three function models fit on example estimates produced by *averaged grouping* with $k = 9$ for the *seeds* dataset . . . 22

3.3 Difference of function shape for the sigmoid model when using weighting or the no-information rate . . . 23

4.1 The estimated kernel density for a grid with one positive and two negative instances; lower Z value indicates lower certainty for the class assignment 30

4.2 Visualizations of the datasets checke1, 2dData, seeds and a downsized version of abalone [8, 9, 25, 29].
The illustration of seeds and abalone was done using an implementation of t-SNE [39] . . . 32

4.3 Average mean errors for the different active learners and datasets using the exponential model. The darker colors of a bar mark the errors of later learning stages (bright -> dark: $[3, 7]$, $[8, 15]$, $[16, 30]$ training set size) . . . 34

4.4 Average mean errors for the different active learners and datasets using the sigmoid model. The darker colors of a bar mark the errors of later learning stages (bright -> dark: $[3, 7]$, $[8, 15]$, $[16, 30]$ training set size) . 35

4.5 Average mean errors for the different active learners and datasets using the linear model. The darker colors of a bar mark the errors of later learning stages (bright -> dark: $[3, 7]$, $[8, 15]$, $[16, 30]$ training set size) . 36

4.6 Average mean errors for different datasets with random sampling. The darker colors of a bar mark the errors of later learning stages (bright -> dark: $[3, 7]$, $[8, 15]$, $[16, 30]$ training set size) 37

4.7 Average squared errors for the different active learners, function models and datasets 38

4.8 Kernel density estimations from the learning process of PAL for checke1 at 3, 8, 16 and 21 training instances 39

4.9 Average Kullback-Leibler divergence for selected methods 40

4.10 Left: Average computation times for the estimators. Right: Histogram of computation time for pathSuper 40

List of Tables

3.1 List of symbols and their meaning 16

4.1 Summary of all estimators evaluated in the tests 27

4.2 Function-specific parameters for model fitting 33

List of Algorithms

1 Pseudocode for capped sub-sampling 17
2 Pseudocode for path sub-sampling 19
3 Pseudocode for path-superset sub-sampling 20

1. Introduction

The field of machine learning has changed quite a bit in the past couple of decades. While it was difficult in the past to gather enough data on some characteristic to create reliable automatic classification, todays processing speed and memory capacities as well as the global connectivity via the Internet allow for large databases available for use. However, often only the part of the data which will be used to predict the characteristic is abundant, which is called *features*. The other part, in case of a discrete characteristic going by the name *class label*, has to be assigned by an annotator. This can result in errors and be costly, especially if the annotator is a human. Using the field of speech recognition as an example, spoken words are easy to obtain from videos, radio or phone calls, what words exactly were spoken is usually unknown. To be able to use the data, a human would have to listen to it and write the correct words down, which is a boring and time-consuming, but necessary, work.

Due to this, a class of algorithms called *active learners* has formed to help minimize the work necessary. As a rule of thumb, the more annotated data is used for the classification, the better it will perform, i.e. less misclassifications occur. But it is important which part of the data has been selected, as some instances, the combination of features and class label, are more informative as others. Thus, annotating the useful instances first and the rest later or not at all minimizes the impairment of the classifier's performance for equal effort. In order to stop the annotation when it is no longer sensible or wanted, we need some way of estimating the current performance. Traditionally, this is done one of three ways: either some part of the annotated data is set aside for so-called holdout testing, which is unwanted since it wastes precious annotated data, some kind of partitioning to create artificial test sets is done, or the classifier's performance in the past in conjuncture with a suitable function model is used to extrapolate to the current situation. The second option struggles with systematic deviation of their estimations from the true performance, while the third heavily depends on the model chosen as well as a number of already present estimates.

In this thesis I will investigate the properties of methods combining partitioning as well as information about the learning process. For this, subsets are created from the data used for training. These are then seen as individual, smaller training sets and build the basis of classifiers themselves, with the remaining data serving as a test set. This results in a number of performance estimations for training sets of various sizes, which can be seen as labeled instances for a different learning problem and are used to learn a model predicting the performance development of the original classifier. Applying the model to the current classifier state described by its training set size then results in a performance prediction. The goal is to evaluate the potential bias and error spread of this method family in comparison to the state-of-the-art estimators k-fold cross-validation and .632+ bootstrapping using different active learners, datasets and model types.

The next chapter gives an overview of classification and active learning in general and touches on currently existing ways of estimating the performance of a classifier. After that, chapter 3 explains the concept of the methods to be evaluated and its variations. After that follows the *Evaluation*, first presenting the settings the testing and later its results. The end is marked by the conclusion, summarizing the results and providing an outlook for possible future work in this sector.

2. Background and Related Work

The main focus of this work is the estimation of classifier performance in the special case of an active learning scenario. To help better understand the problem setting, we will give a brief overview of the concepts of active learning as well as an in-depth look of existing methods for said estimation. We also establish parts of the notation that will be used for the remainder of this work.

2.1 Classification

Many problems to be solved encompass the differentiation between certain ”classes” to which their inputs can be assigned. To explain the concept of **classification**, we assume the example of a thermostat. Its purpose is to monitor the temperature in a certain area and fire up a heating unit to raise the temperature if necessary. But for that to happen, the thermostat has to *classify* the measured temperature in the categories ”too cold” and ”warm enough”. In this special case a simple threshold usually suffices. Generally, however, a *classifier* C is defined as a mapping of a *feature vector*, also called an *instance*, to its corresponding *class label* $y \in Y = \{y_1, ..., y_m\}$. It is convenient to describe said vector as an n-tuple of *feature values*: $\vec{x} = (x_1, ..., x_n)$ with $x_i \in F_i$ $\forall i = \{1, ..., n\}$ and F_i as a *feature*. The goal of a classifier is to approximate the underlying ”true” association of a feature vector to its class label $f(\vec{x}) = y$, which can be written as

$$C : F_1 \times ... \times F_n \to Y \tag{2.1}$$

or $\hat{f}(\vec{x}) = \hat{y}$ [32]. This definition makes it clear that classification is not restricted to single-variable problems. To stick with our example of a thermostat, you may want to consider the humidity, temperature outdoors or whether any windows are open to decide if heating is necessary. Closely related are *regression problems.* While a classifier works with discrete class labels, regression uses a continuous output space.

To obtain a classifier for a specific problem, one makes use of what is called a *learning algorithm A*. In a process called *training* a set of instances, the *training data* $X_T =$

$\{(\vec{x}_1, y_1), ..., (\vec{x}_n, y_n)\}$, is taken and such a mapping C using an optimization criterion is created. The training set is a subset of a distribution of all possible data and should be independently and identically distributed. The whole data set can be modeled as a bivariate distribution with the feature vectors and class labels as random variables: $(\mathbf{X}, \mathbf{Y})$ [32]. While all learning algorithms are being fed feature vectors to create a classifier, they can be separated into three categories, depending on their requirement for labeling:

- **Supervised:** This type of algorithm requires all of their input data to be labeled as well as a list of all possible class labels. While it may seem like the best option, its drawbacks include the potential cost of the labeling. If the correct class labels are not inherently known, usually a human is required to assign the correct labels manually. With unlabeled data readily available for problems like speech recognition, this is a very expensive part of the process. Another issue is the propagation of errors; any mistake made during labeling is carried over into the modeling of the classifier.

- **Unsupervised:** Instead of requiring all data to be labeled, unsupervised algorithms ignore class labels completely. This saves the cost of labeling the data, but many algorithms require some sort of tuning parameters (e.g. the cardinality of Y and shape of underlying probability distribution).

- **Semi-supervised:** As a compromise of the two extremes, semi-supervised techniques operate with a mix of labeled and unlabeled data. This way it seeks to combine the low effort for labeling with the advantages of supervised learning [45].

Regardless of their category, all learning methods make assumptions about the distribution of the data: feature vectors close to each other tend to belong to the same class and, consequently, data points are likely to form group-like structures.

2.2 Active learning

As a subgroup of semi-supervised learning, active learners try to optimize the performance of a classifier when using a fixed amount of class labels. To achieve this, they make the training an iterative process, selecting one or more instances for labeling that the believe will benefit the performance the most in each iteration. While more exist, we will touch on three of them here:

2.2.1 Random sampling

This active learner is the baseline for all other ones. It does not show a preference for any instance, hence its name, making at is good or bad as not using active learning at all.

2.2.2 Uncertainty sampling

The *uncertainty* of a classifier with regard to an instance describes how sure it is about the class label it would assign. This can be measured in various ways, three of which are:

- **Least confidence**: This measure assumes that the instance for which the classifier produces the lowest probability estimate for its most likely class label $\hat{y}$ has the highest uncertainty u [11]:

$$u_{LC}(\vec{x}) = 1 - P_c(\hat{y}|\vec{x}) \tag{2.2}$$

- **Smallest margin**: Here the difference in probability between the *two* most likely class labels y_1, y_2 for an instance is used as the uncertainty measure [34]:

$$u_{SM}(\vec{x}) = P_c(y_1|\vec{x}) - P_c(y_2|\vec{x}) \tag{2.3}$$

- **Maximum entropy**: Instead of only using the probability of a few class labels, the entropy takes all of them into account [12]:

$$u_{ME}(\vec{x}) = -\sum_{y} P_c(y|\vec{x}) \cdot log_2(P_c(y|\vec{x})) \tag{2.4}$$

It should be noted that, while they possibly select different instances in multi-class problems, the first two are numerically identical for dichotomous data and all three will select the same instance [44]. In any case, they all assume what is called a *decision boundary*, an entity which spatially separates groups of instances sharing the same class label. This can problematic if multiple clusters of the same label exist, as they are likely to be regarded as if they would belong to a different group.

2.2.3 Probabilistic Active Learning (PAL)

PAL takes a different approach. It takes a look at each available instance's surroundings and computes the density of unlabeled as well as the amount of already purchased instances. The latter combined with the portion of instances sharing their class label with the original instance is denoted as *label statistics*. Using this, assuming the instance's class label and the overall probability of this class in its neighbourhood was to be known, an optimal selection with regard to the classifier's performance could be made. But since those two quantities are generally unknown, PAL considers them as random variables. This enables the computation of an expected or probabilistic performance gain for each unlabeled instance. When weighted with the instance density in its neighbourhood, it allows an statistically optimal selection for the performance [25]. Figure 2.1 show the probabilistic gain for the remaining instances, with the highest

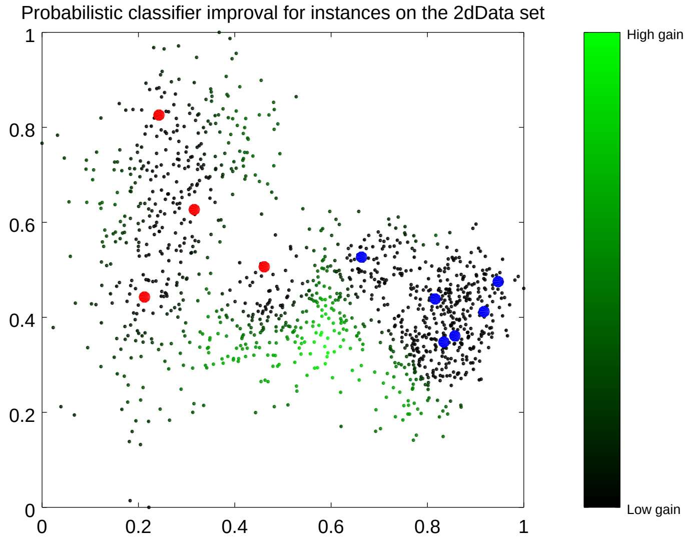

Figure 2.1: PAL illustration showing the weighted probabilistic gain (normed on $[0, 1]$). The pgain is highest in areas where no labels have been purchased yet

gain in the area with a low label density which is also the border region between both classes.

Figure 2.2 shows the state of a Parzen-Window classifier (see Section 4.3.4) with 16 purchased labels for random sampling, uncertainty sampling and PAL on a 4x4 checkerboard-like dataset with every second field without instances. This structure can be seen fairly well with PAL, which correctly spreads out its purchases over the entire board, while the classifications for random sampling at least resemble its true structure. Uncertainty sampling, however, only purchased labels at the first decision boundary it found and did not bother to look beyond that, resulting in a model that looks nothing like a checkerboard.

2.2.4 Model Retraining Improvement and Monte-Carlo-Sampling

Introduced in [16], **Model Retraining Improvement (MRI)** is originally intended as a statistically optimal criteria for instance selection in active learning. It does so by selecting the instance which reduces the future loss the most. However, for this it needs an (ideally unbiased) estimate of the future and the initial loss, which the algorithm itself does not provide. Instead, the paper provides a theoretical background for potential future estimators.

Ray and McCallum [33] also presents a method to estimate current and future loss of a classifier in an active learning setting, similarly to MRI. They, however, use *Monte-Carlo-Sampling*, resulting in a process comparable to the techniques explained in the next section.

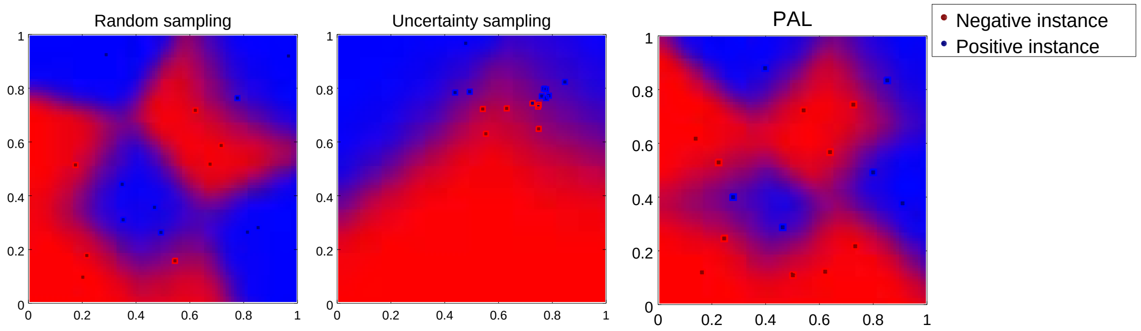

Figure 2.2: Classifier state for random and uncertainty sampling as well as PAL for 16 purchased labels on checke1

2.3 Generalization Performance

When a learning algorithm is used to induce a classifier, an obvious question is how well said classifier performs or how well it approximates the true target function $f(\vec{x})$. This can be used to select an algorithm when multiple are available or to simply get an estimate on how often the classifier will misclassify data.

To facilitate the calculations further down a closer look at the training set X_T is helpful. As earlier stated, the individual instances are supposed to be independently and identically distributed, which means the set can be seen as a random variable. Now the probability of a specific set depends on the probabilities to draw each of the instances and, in case of supervised and semi-supervised learning, their associated labels. Due to their independence, we can write it as

$$p(X_T) = p(\vec{x}_1, y_1) \cdot \ldots \cdot p(\vec{x}_n, y_n) = \prod_{i=1}^{n} p(\vec{x}_i, y_i) \tag{2.5}$$

A similar formula applies for unsupervised learning. [32]

2.3.1 (Expected) prediction error, training error and loss

To effectively evaluate the performance of a classifier it is important to quantify when it is mistaken. The *loss function* $L(f(\vec{x}), \hat{f}(\vec{x}))$ describes the error a classifier makes for a specific feature vector. A popular loss function, especially for two-class problems, is *0-1-loss*:

$$L(f(\vec{x}), \hat{f}(\vec{x})) = \begin{cases} 0, & \text{if } f(\vec{x}) = \hat{f}(\vec{x}) \\ 1, & \text{else} \end{cases} \tag{2.6}$$

For regression problems squared or absolute error loss are more effective, since equal function values are improbable in a continuous output space. It seems intuitive to use the feature vectors already used for training again for the performance evaluation, especially since they are already labeled and with that $f(\vec{x})$ for them known. Utilizing the loss function from above, the in-sample error or *training error* [19] on the training set $X_T = \{(\vec{x}_1, y_1), ..., (\vec{x}_n, y_n)\}$ is

$$Err_T = \frac{1}{N} \sum_{i=1}^{n} L(y_i, \hat{f}(\vec{x}_i)) \tag{2.7}$$

If used with the 0-1-loss function, $1 - Err$ is also known as *accuracy*:

$$acc = 1 - err = \frac{|CorrectClassifications|}{|TotalClassifications|} \tag{2.8}$$

Unfortunately, using the measure to judge a classifier produces a side effect. A learning algorithm which creates a complex classifier that perfectly classifies all training instances will yield a training error of zero. If used on different data points from the same data set, however, an increase in misclassifications will be noted, likely more than for a less complex classifier with a few misclassifications on training data. This phenomenon is known as *overfitting*, resulting from the memorization of X_T by the classifier while a generalization onto the whole data set was wanted [13].

A more general error measure is the *prediction error*. It draws independent samples from the data distribution $(\mathbf{X}, \mathbf{Y})$ and uses these to examine the loss of a classifier by calculating the expected value over all possible realizations:

$$Err_S = E_{(\mathbf{X},\mathbf{Y})}[L(\mathbf{Y}, \hat{f}(\mathbf{X}))|X_T] \tag{2.9}$$

[32]. This approach bears a problem: the distribution of our data is usually unknown, hence the need for a classifier. In general, the prediction error cannot be computed directly, thus the need for estimators arises. An important aspect of the prediction error is the dependence from the fixed training set. This allows for the performance estimation of an already trained classifier. A different approach takes away that dependence and instead calculates the expected error for all possible training sets $Err_E = E_{X_T}[Err_S]$, which equals

$$Err_E = E_{(\mathbf{X},\mathbf{Y})}[L(\mathbf{Y}, \hat{f}(\mathbf{X}))] \tag{2.10}$$

Using the probability of a specific training set from Equation 2.5, it can also be written as $Err_E = \sum_{X_T} p(X_T) \cdot Err_S = \sum_{X_T} Err_S \prod_{i=1}^{|X_T|} p(\vec{x}_i, y_i)$. Here the performance of the algorithm that creates the model $\hat{f}(\vec{x})$ is evaluated, which is no longer of use for the analysis of a trained classifier but can guide the selection of a preferable algorithm, accounting for all possible training sets. Note that we still require knowledge of the underlying distributions for a direct evaluation, realistically making an estimator necessary as well [19].

2.3.2 Bias and variance

In the previous sections we portrayed classifier as a mapping $\hat{f}$ of input values $\vec{x}$ to class labels y. Their general task is, however, to assign *probabilities* for each of the possible class labels. Given our random variable $\mathbf{Y}$ for the class labels and a fixed value $\vec{x}$ of our input variable $\mathbf{X}$, $P(\mathbf{Y} = \hat{y}|\vec{x})$ represents the probability that $\mathbf{Y}$ realizes as the value y given our input. Some classifiers, like decision trees, assign a non-zero probability only to one class label given an input, leading to the possibility of being written as a function. In fact, since in practice a definite class label is needed, most classifiers will pick the class label which maximizes said probability, resulting in function-like behaviour anyway [23].

Another assumption that doesn't necessarily hold was that an error-free target function $f(\vec{x})$ exists. It is entirely possible for our target function to be *noisy*, that is to randomly vary from its true value. In turn, this *variance* leads to blurry class assignments. Thus, we only get a probability $P(\mathbf{Y_T} = y|\vec{x})$ instead of a sharp mapping. Note that the class variable here is different from the one in the previous paragraph; they are conditionally independent for our target distribution and a fixed. Now with this probabilistic notation, it is possible to *decompose* the expected prediction error from the previous section into three components:

$$\begin{aligned} Err_E &= \sum_{\vec{x}} P(\vec{x}) \left(\sigma^2(\vec{x}) + bias^2(\vec{x}) + variance(\vec{x})\right) \\ &= \frac{1}{|X|} \sum_{\vec{x}} \left(\sigma^2(\vec{x}) + bias^2(\vec{x}) + variance(\vec{x})\right) \end{aligned} \tag{2.11}$$

The bias and variance express by how much our classifier differs systematically and at random for a given data point from the true value. σ^2 is the variance of the noise distribution that the target may or may not have; it is the irreducible error that any classifier will always make. The need for the probability of $\vec{x}$ is dropped under the assumption of equal likelihood for all data points [23]. Ideally you would want to minimize both the bias and the variance of your classifier to achieve a low prediction error. Unfortunately, as the model complexity increases to accommodate for more subtle structures in the training data its bias will decrease but the variance will increase. This dilemma is known as the *bias-variance-tradeoff* [26].

2.3.3 Classifier-based estimators

As stated section 2.2.1, the training error of a classifier usually underestimates the true prediction error Err_S. To calculate the desired error, two options are available. Either a direct estimate is taken, usually with dedicated test sets, which will be called *out-of-sample error* since it uses different data points than the training error, or the difference between training and true error is estimated and then added to the training error, also called *optimism*. This section briefly introduces three methods to estimate the optimism: *Akaike information criterion* and *Bayes information criterion*.

The **Akaike information criterion**(AIC) in its original form was introduced by [2] in 1973. It is defined as a function of a model's likelihood and complexity λ: $AIC = -2 \cdot log(likelihood) + 2\lambda$ [6]. If instead of a 0-1-loss the log-likelihood-loss is used, a quantity that uses the logarithm of the likelihood of a model given the input, and a linear model with λ parameters assumed, the AIC can also be written as

$$AIC = Err_T + 2\frac{\lambda}{n}\sigma^2 \tag{2.12}$$

with n as the number of training instances and σ^2 the variance of the target model [19].

A very similar estimate is the **Bayesian information criterion**(BIC). Presented by [35] as an alternative to AIC, its definition is similar: $BIC = -2 \cdot log(likelihood) + \lambda log(n)$. It is asymptotically equal to AIC for the size of the training set, but with a steeper penalty for complex models due to the exchange of the factor 2 with $log(n)$ [41]. Using the same assumptions as for AIC, it can be calculated as

$$BIC = \frac{n}{\sigma^2}[Err_T + log(n) \cdot \frac{d}{N}\sigma^2] \tag{2.13}$$

While both criteria are theoretically solid, they are impractical for use in this work; both assume fairly simple model families and restrict the use of loss functions [19].

The third estimator for the optimism is based on the **Vapnik-Chervonenkis(VC) theory**. In their publication [40] they introduce an algorithm-specific quantity called VC-dimension which is defined as the number of data points a classifier can separate, regardless of their position and class label. Based on this, they derived an upper bound for the optimism of the training error:

$$sup|Err_S - Err_T| \leq 2\frac{ln(\frac{2|X_T|}{h})}{l/h} \tag{2.14}$$

with h as the VC-dimension. The derived bound has some restrictions, however; it is only valid for large training sets and requires knowledge of the VC-dimension. Unfortunately, analytical solutions are known only for a few algorithms and it is not given that it is constant with regard to the training set size [7].

2.3.4 Cross-Validation

As the training error overestimates the classifier performance due to the same samples from the data being used for training as well as evaluation, a different approach is to use separate data for testing. Using this data set called *hold-out set* gives a direct estimate of the prediction error. A common split is to use two thirds of the available (labeled) data as training data and the rest for testing. However, since labeling data can be expensive, often only little labeled data is available. In that case, holding out a third of it may significantly reduce the classifier's performance. An alternative approach is known as *cross-validation* in which the classifier is trained with the full training set. For the performance evaluation the classifier is then retrained with a part of the data, the rest is used for testing [22].

K-fold cross-validation is the most simple form of cross-validation. Given a set of labeled data X_T, k new sets X_k are created, each with size n. For each of the k sets a classifier is trained with $X_T \setminus X_k$. X_k then serves as the test set on which the classifier is evaluated, resulting in k estimations of the performance to be expected of the classifier trained with the whole set X_T:

$$\widehat{Err}_{S,i} = \frac{1}{n} \sum_{j=1}^{n} L(y_{i,j}, \hat{f}(\vec{x}_{i,j})) \tag{2.15}$$

with $i = \{1, ..., k\}$ [22]. These estimations can be seen as samples of a performance distribution; examples can be uniform (simple average) or beta distribution [25]. As a special case of k-fold cross-validation, **leave-one-out** cross-validation works with test sets of size one. To ensure similarity to the true data distribution, the folds are usually *stratified*; this way the ratio of class labels in X_T is kept also in the test sets.

More expensive variants are **complete** and **leave-pair-out** cross-validation. The former uses every possible two-set split of X_T as basis of the estimation, while the latter only regards the possible pairs of data points. Complete cross-validation is rarely used due to its computational complexity [22], while leave-pair-out seems to strike an acceptable mix of effort and accuracy [30].

Leave-p-out cross-validation generalizes the other variants. Here, every subset of size $p \in \{1, ..., |X_T| - 1\}$ is used as a test set, resulting in $\binom{|X_T|}{p}$ estimates which are then averaged [4]. Leave-one-out and leave-pair-out are equivalent to leave-p-out with $p = 1$ and 2, respectively.0

An algorithm specifically designed to work with iterative labeling was introduced by [7] under the name of *adaptive incremental k-fold cross validation*. It performs k-fold cross-validation after each increase of the training set size and stops when a certain performance threshold is reached. [1] perform an empirical study of the behaviour for different cross-validation techniques as well as *bootstrapping*, which will be described in the next section. Their findings indicate leave-pair-out and k-fold with $k = 5$ or 10 with averaged results as the most robust approaches. They also find them to be unbiased performance estimators, which is also stated in [22]. [32] clarifies that this only holds true for the performance estimation of classifier trained with $n - \frac{n}{k}$ instances. The estimation of classifier performance trained with n samples is afflicted with a *positive error bias*, i.e. the value of the estimated error rate is higher than the true error rate. [15] reinforce this argument with experiments and provide assessment of the variance as well.

2.3.5 Bootstrapping

While closely related to cross-validation, **bootstrap** takes a slightly different approach. Instead of splitting X_T into mutually exclusive sets, b sets of equal size n are created and filled by random sampling of elements from X_T. These are sampled with replacement, meaning can be drawn multiple times into the same set. The classifier is then trained on every bootstrap set X_i^b and tested against the original set X_T [22]:

$$\widehat{Err}_i^{boot} = \frac{1}{|X_T|} \sum_{j=1}^{|X_T|} L(y_j, \hat{f}_i^b(\vec{x}_j)) \tag{2.16}$$

Using the whole set X_T as a test set, however, has some unwanted side effects. Because the bootstrap set used to train the classifier and X_T overlap, the classifier's performance is estimated with samples it has already seen in training, leading to a *negative error bias* for the estimated error rate. To avoid this, the bootstrap trained classifiers are only tested on samples which are not part of their training set, known as **leave-one-out bootstrapping (LOO)** [5]:

$$\widehat{Err}_i^{LOO} = \frac{1}{|X_T \setminus X_i^b|} \sum_{(\vec{x},y) \in X_T \setminus X_i^b} L(y, \hat{f}_i^b(\vec{x})) \tag{2.17}$$

A different bootstrap estimator was introduced by [14]. While vanilla bootstrap is biased downwards with regard to the estimated error rate, its leave-one-out version goes over the top. The reason for this is similar to the problems of cross-validation: Since the classifier is trained with less instances than available for the estimation, it lacks knowledge that a classifier learned on the full training set has, leading to a worse prediction performance and thus underestimating the performance. As proven by Efron in the same publication, the fraction of unique instances from X_T in a bootstrap set is expected to be $1 - e^{-1} \approx 0.632$. By seeing bootstrapping actually as an estimator for the optimism, similar to AIC and BIC, he derives the so called **.632 bootstrap**:

$$\widehat{Err}^{.632} = 0.368 \cdot Err_T + 0.632 \cdot \frac{1}{b} \sum_{i=1}^{b} \widehat{Err}_i^{LOO} \tag{2.18}$$

It factors in the training error to lift up the otherwise too conservative estimate of the performance [14].

Building on that, [15] developed a modified .632 bootstrap under the name of **.632+ bootstrap**. As their experiments showed, the .632 method has problems estimating the performance of an extremely overfit classifier. To account for this, they use two the two quantities *no-information error rate* and *relative overfitting*. The no-information error rate $\hat{\gamma}$ is defined as as the error rate of a classifier when no dependence exists between input and output data, i.e. $\mathbf{X}$ and $\mathbf{Y}$ are independent. Relative overfitting $\hat{R}$ then expresses how close the classifier is overfit to the no-information error rate:

$$\hat{R} = \begin{cases} \frac{\widehat{Err}^{LOO} - Err_T}{\hat{\gamma} - Err_T}, & \text{if } \widehat{Err}^{LOO}, \hat{\gamma} > Err_T \\ 0, & \text{else} \end{cases} \tag{2.19}$$

The .632+ error rate is then defined as

$$\widehat{Err}^{.632+} = \widehat{Err}^{.632} + \Big(min(\widehat{Err}, \hat{\gamma}) - Err_T\Big) \cdot \frac{0.368 \cdot 0.632 \cdot \hat{R}}{1 - 0.368 \cdot \hat{R}} \tag{2.20}$$

Part of the assumptions behind this estimator is that leave-one-out bootstrapping has the correct expected error rate value in the case of independence of **X** and **Y** and a class probability of 0.5 in a 2-class problem, whereas the .632 does not [15]. However, [42] show in their paper about different error rate estimators that leave-one-out slightly underestimates the error. This stems from the likely inequality of class labels amongst the drawn instances. They show that in such a case, on average, one class is 22% larger given a random sample. The nearest-neighbour classifier, as assumed by Efron and Tibshirani, assigns an instance the same class label as the closest instance. Since the assumption for the data was that no correlation between feature vector and class label exists, this assignment is random. However, due to the imbalance of class labels, the assignment probabilities are unequal, leading to more classifications for the larger class. In turn, leave-one-out cross-validation will place its error estimate slightly below 0.5. For larger sample sizes this effect is more pronounced; the average class size difference scales with the square root of the sample size. Unfortunately, no experiments with .632+ were performed to evaluate the influence on its estimation performance.

2.4 Learning Curves and Regression Models

The error measures described in section 2.2.1 give an (theoretically) accurate idea about a classifier's performance for either a specific or all possible training sets. In the case of, for instance, active learning, the learning algorithm isn't just fed a single training set. Instead, instances are added iteratively and a new classifier gets trained each round. For this, a representation of the algorithms progress in terms of the resulting classifiers' error rates seems desirable. Originating in the field of psychology, it describes a collection of models to assess the level of comprehension in an individual over the time of exposure or number of examples [43]. Similarly, in the context of machine learning it describes a function mapping the size of the training set to a measure of performance; a commonly used measure is accuracy [31].

As a known functional dependency between accuracy and training set size would enable an easy lookup of the to-be-expected error rate for a certain amount of labeled data, attempts have been made to find a fitting function model.

2.4.1 Function families

The typical form of a learning curve is a steep increase in accuracy when few examples have been presented, leveling more and more for an increased training set, converging towards a maximal accuracy [17]. An example of such a curve can be seen in figure Figure 2.3.

Functions families fitting this description are **power** [17, 38], **logarithmic** and **exponential** [38]. These papers provide an overview for the fitting of the function types.

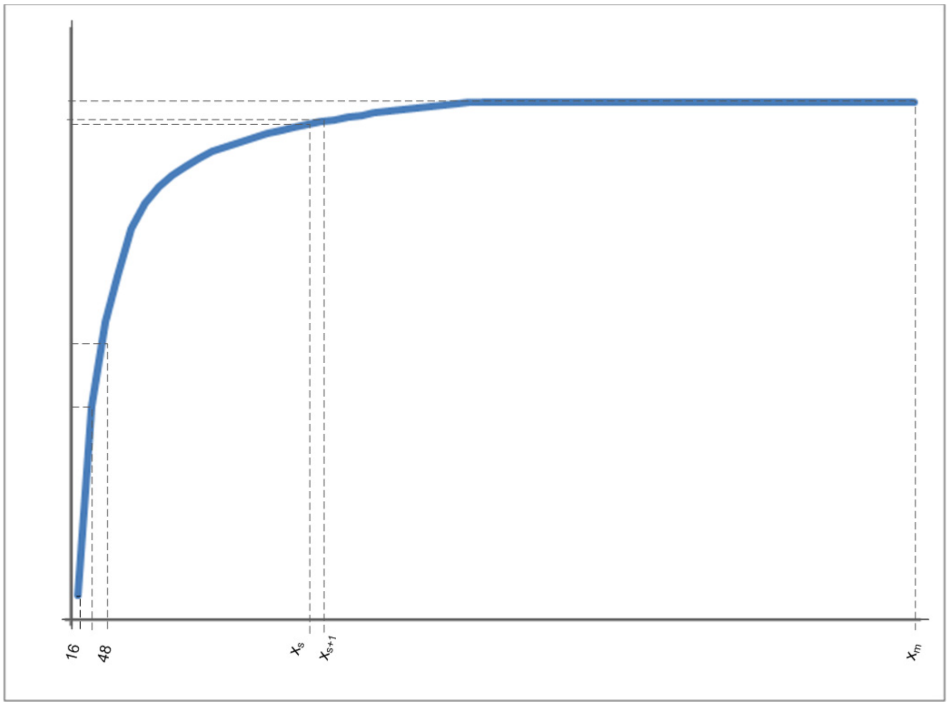

Figure 2.3: General appearance of a learning curve. Training set size as x-axis, accuracy as y-axis [17]

[38] uses linear fitting with least squared error as the target and the resulting correlation coefficient as a measure of fitting for four different function models and four classifiers. The power law $f(x) = a + b \cdot x^c$, with a, b, c being regression parameters, performed best on most data sets tested and is also the basis for a proposed method of performance prediction in [17]. It has to be noted, though, that in a fourth of all cases outperforms a linear model the more complex models, mostly on the same data set. This may hint at a data set dependency for the correct curve model.

A related approach is described in [10]. Here, instead of modeling the prediction error directly, an exponential model is assumed for the optimism of the training error: $Err_S - Err_T = \frac{2b}{|X_T|^\alpha}$ and $Err_S + Err_T = 2a$. Using already acquired (true) error rates, the parameters are estimated from the gradient and amplitude of the data after being transformed into a logarithmic scale and fitted linearly. Extrapolating the function then gives an estimate of the expected error rate for a given $|X_T|$.

2.4.2 Maximum-likelihood regression

Similar to the extrapolation of fitted learning curves, the *maximum-likelihood-regression* presented in [21], aims to find the most likely model to explain the given data. Based on this, the model is then extrapolated to obtain an estimate.

3. Modeling performance estimation as a learning process

The currently existing performance estimators recalled in the previous chapter are made up of two groups. The first kind only uses information available for the already trained classifier; the cross-validation and bootstrapping family belong to it. The other one collects information during the classifier's training, using it to fit and extrapolate a learning curve. How that information is obtained differs, common practices include the use of holdout test sets and k-fold cross-validation [17].

The methods evaluated in this thesis build on both approaches. Generalizing the extrapolating approach, estimating the performance of a classifier based on individual estimates is a type of learning itself, falling into the category of *regression* since performance is not discreet. But instead of using the classifier's journey, the combined estimator simulates different training sets that could have been an earlier state of the classifier, i.e. the subsets of the current training set. To help illustrating this method, the following scenario is assumed: a dataset composed of both labeled and unlabeled data $\mathcal{D} = \mathcal{L} \cup \mathcal{U}$ with $\mathcal{L} = \{(\vec{x}_i, y_i) : i \in \{1, ..., n\}\}$ and $\mathcal{U} = \{(\vec{x}_i, .) : i \in \{n+1, n+2, ...\}\}$, $y \in \{0, 1\}$, serves as the basis for an active learner, which selects a training set $X \subseteq \mathcal{L}$ of size $k \leq n$. This set is used to train a classifier $c_X : \vec{x} \mapsto y$. We now would like to know the accuracy $acc_c(X, \mathcal{D})$ the classifier has on the entire dataset.

3.1 Performance estimation on training sub-sets

In order to obtain the performance estimates for the classifier trained on training subsets, *leave-p-out (LPO) cross-validation* with $p \in \{1, ..., k-1\}$ is used. As p is the number of omitted and thus test instances, the training subsets $S_i^{k-p} \subset X$ are of size $|S_i^j| = j$ and $i = \{1, ..., \binom{k}{j}\}$, resulting in $\binom{k}{p}$ accuracy estimates for each subset size. The corresponding test sets are $T_i^{k-p} = X \setminus S_i^{k-p}$. While LPO is a pessimistic estimator

Symbol	Description
$\mathcal{D}$	Whole dataset
$\mathcal{U}$	Unlabeled part of $\mathcal{D}$
$\mathcal{L}$	Labeled part of $\mathcal{D}$
X	Current training set
k	Size of X: $k = \|X\|$
$\vec{x}_i$	Instance of X: $\vec{x}_i \in X \forall i = \{1, ..., k\}$
y_i	Class label of instance $\vec{x}_i$
c_X	Classifier trained with set X
$acc_c(A, B)$	True accuracy of classifier c trained with set A on the instances B
$\widehat{acc}_c(A, B)$	Estimate of $acc_c(A, B)$
S_i^j	ith subset of X with size j
$\check{S}$	Selected subsets of X from capped sub-sampling
$\tilde{S}$	Set of paths (n-tuples of size $k-1$)
$T, \check{T}, \tilde{T}$	Test sets $X \setminus S_i^j$ for estimation
Y_m	Set of tuples containing a subset estimate and the subset's size; input of the fitting algorithm
$f^m(j)$	Function fitted using Y_m depending on j as the classifier's training set size
H_i	Holdout test sets
$\mathcal{X}^{\Vert}$	Set containing all training sets of size k used for testing
K	Kernel for KDE
h	Bandwidth for KDE

Table 3.1: List of symbols and their meaning

when applied to a classifier with training set size k, i.e. it systematically places the accuracy lower than its true value, it is *unbiased* for a classifier with a training set of size $k-p$ [32]. This leaves us with $2^k - 2$ estimates in total, each of them unbiased, but at the cost of exponential time complexity; some sort of selection may be reasonable to reduce it.

Another option with regard to the estimation is *bootstrapping*. It offers a little more variety, as different types like *naïve*, *leave-one-out* and *.632* are available. Also, individual estimates are more expensive to compute, as multiple bootstrap samples are first created. Also, there is no real equivalent to LPO for bootstrapping, meaning that the estimation looks slightly different: for *leave-one-out (LOO) bootstrapping*, an estimate for a subset S_i^j and a corresponding bootstrap sample is actually the average of j estimates, holding out each of the j instances once and testing the resulting classifier on the hold-out instance. That is how bootstrapping was defined originally, anyway; however, the instances in T_i^j not used for training would be left out. As they are not part of the training, they may as well be used as a test set. Problematic is that .632 bootstrap relies on a ratio of expected pessimism and expected optimism of LOO's estimate and the classifier's training error respectively. This is based on the probability of one instance not occurring in n draws with replacement, i.e. how likely it is that a bootstrap sample does not contain a given instance. Since the instances in X S_i^j cannot possibly occur in the bootstrap sample, which is drawn from S_i^j, the bootstrap estimate should be less pessimistic and the ratio may not hold anymore.

3.1.1 Sub-sampling of fitting points

Regardless of which estimation technique is used, the complexity still scales exponentially with the training set size k. To reduce the amount of estimates for the model fitting process, some sort of selection has to occur. In this section, three sub-sampling strategies are explored. However, it is to be kept in mind that reducing the information available naturally has some drawbacks, including an expected higher variance and, if not done properly, an added bias. Also, the sampling may influence the fitting itself, potentially inflicting additional penalties to the robustness.

A simple, yet effective method which will be called *capped sub-sampling* is to cap the number of estimates. Possible options are to either impose a fixed, hard cap for all training set sizes k, or to use a polynomial dependent on it, e.g. k^2. A potential pitfall is the selection of the remaining subsets: selecting either randomly over all possible subsets or from pools for each subset size $k - p$ with sizes proportional to $\binom{k}{p}$ prevents unintentional importance assignment to subset sizes. The computation of this reduced set $\check{S} \subseteq S$ is illustrated in 1, the corresponding test sets $\check{T}_i^j = X \setminus \check{S}_i^j$.

```
$X = \{(\vec{x},y_1), ..., (\vec{x}_k, y_k)\}$                    ▷ Current training set
$numOfSamples \leftarrow ...$

$S \leftarrow powerset(X)$                                        ▷ Compute all possible subsets

$\check{S} \leftarrow \emptyset$
for $j \leftarrow 1$ to $k - 1$ do
    for $i \leftarrow 1$ to $numOfSamples$ do
        $currSample \leftarrow drawWithoutReplacement(S^j)$      ▷ Get a random subset of size j and remove it
        $\check{S} \leftarrow \check{S} \cup currSample$
    end for
end for
```

Algorithm 1: Pseudocode for capped sub-sampling

A related approach exports the computational cost to the fitting process. Instead of selecting multiple subsets S_i^j per size once and using them as a basis for one model, this sub-sampling creates multiple models and selects only one subset S_j per j. Formally, we have a tuple $\tilde{S}_i = (s^1, ..., s^{k-1})$ with $i \in \{1, ..., r\}$ and $s^j \in S^j$, which will be called a *path*. Their respective test sets are $\tilde{T}_i = (X \setminus s^1, ..., X \setminus s^{k-1})$. The S_j can be drawn with or without replacement, although the latter may lead to a lower variance, as seeing the same constellation multiple times does not add information, whereas a different one does. The parameter r is up to choosing, with an upper limit of $\prod_{i=1}^{k-1} \binom{k}{i}$ as the number of combinations for drawing with replacement; the algorithm for this is depicted in 2 and goes by the name of *path sub-sampling*. This would result in a higher complexity than exponential, namely $O(k!)$. However, accounting for all possible subset combinations may not be necessary, as there are far less unique combinations of accuracy estimations. This is due to the number of test instances available for a given training subset. For example, a classifier trained on a set of size one tested against a set of size three will have four potential test outcomes: either one, two, three or none instances were correctly classified, resulting in an estimated accuracy of $\frac{1}{3}, \frac{2}{3}, 1$ and 0, respectively. As a growth in size of the training set in turn causes a reduction in size of the test set, the amount of potential accuracies $\widehat{acc}_c(S_i^j, T_i^j)$ shrinks from k to 2 for $j = \{1, ..., k-1\}$. Thus, the number of unique combinations would shrink to $k!$.

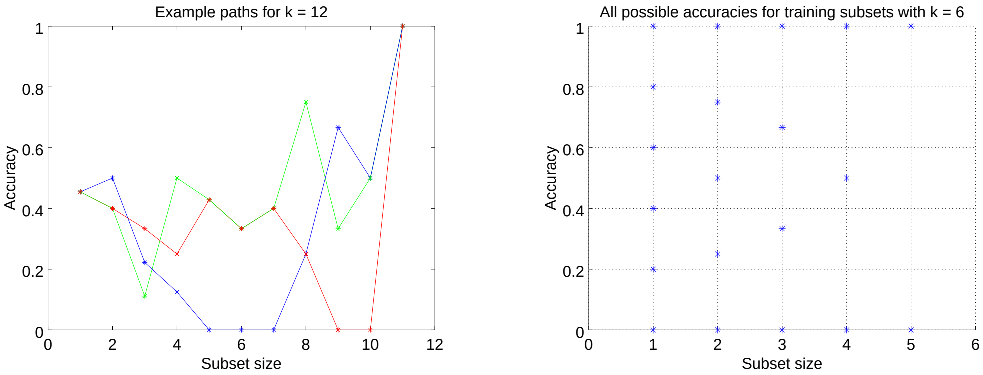

Figure 3.1: Left: Three example paths generated with path-superset sub-sampling. The zigzag shape illustrates the difficulties for the fitting. Right: All possible accuracies for training subsets with $k = 6$

Since the actual realization of the accuracies is not yet known, they would have to be computed first. This means that this approach is inept to reduce the number of needed subsets, but the idea of using multiple models instead of only one is still valid; it may increase the spread of the final estimate $\widehat{acc}_c(X, \mathcal{D})$ due to the reduced information available for each model, but also allows to derive a distribution for the accuracy (see Section 4.1).

Following this, there are other possible restrictions with regard to the selection of the paths $\tilde{S}_i$. As it is, randomly taking subsets of each size allows for instances present in S^j to not be selected for the larger sets $S_{j+1}, S_{j+2}, ...$. However, a classifier trained by an active learner does not usually discard previously selected instances. Thus, it seems logical to restrict subsets of larger size to be supersets of their predecessors $S^j \subset S^{j+1}$;

see 3 for the pseudocode. This again reduces the possible combinations to $k!$, as the sampling from X is now done without replacement. This time, however, the estimates for every subset are not needed because each path is equally likely, meaning that this sub-sampling is applicable even without precomputing the estimates. This assumption only holds in general for random sampling; other active learners may show preferences for some instances which, in their eyes, improves the classifier's performance the most. Due to its restriction of the possible paths this method will be referenced as *path-superset sub-sampling*.

```
$X = \{(\vec{x}_{,}y_1), ..., (\vec{x}_k, y_k)\}$                              ▷ Current training set
$numOfPaths \leftarrow ...$

$S \leftarrow powerset(X)$

$\tilde{S} \leftarrow \emptyset$
for $i \leftarrow 1$ to $NUM\_OF\_PATHS$ do
    $currPath \leftarrow nTuple(k)$
    for $j \leftarrow 1$ to $k - 1$ do
        $currPath_j \leftarrow drawRandom(S^j)$                ▷ Get a random subset of size j
    end for
    $currPath_k \leftarrow i$                    ▷ The same path may get drawn multiple times,
                                                         ▷ but it still has to be in the set
    $\tilde{S} \leftarrow \tilde{S} \cup currPath$
end for
```

Algorithm 2: Pseudocode for path sub-sampling

3.1.2 Extracting learning curves from performance estimates

Now that the subsets have been selected, the individual performances have to be estimated. This requires a classifier being trained on each of them and testing it on the corresponding test set. After that comes the decision of which estimates to use to fit one or more learning curve models, and how. The algorithm doing the fitting takes a set of tuples $Y_m = (a, j)$ with $j \in \{1, ..., k-1\}$ and $a \in [0, 1]$. As a result it outputs a function $f^m : \mathbb{N} \mapsto \mathbb{R}$. There are multiple options available for how the Y_m are chosen:

- **Mix grouping**: In the most simple case, there is only one $Y = \{(\widehat{acc}_c(\check{S}^j_i, \check{T}^j_i), j) : j \in \{1, ..., k-1\}, i \in \{1, ..., |\check{S}^j|\}\}$: every subset estimate is taken and given to the fitter, producing a single function f^1.

- **Averaged grouping**: As originally intended for leave-p-out estimation, the subset estimates are first averaged before passed to the fitting:
  $Y = \{(\frac{1}{|S^j|} \sum_{i=1}^{|S^j|} \widehat{acc}_c(S^j_i, T^j_i), j) : j \in \{1, ..., k-1\}\}$. This also results in only one function, but takes some strain away from the fitting. Problematic may be that the amount of estimates for any subset size j does not vary anymore, withholding information from the fitting.

```
$X = \{(\vec{x},y_1), ..., (\vec{x}_k, y_k)\}$                         ▷ Current training set
$numOfPaths \leftarrow ...$

$S \leftarrow powerset(X)$

$\tilde{S} \leftarrow \emptyset$
for $i \leftarrow 1$ $to$ $NUM\_OF\_PATHS$ do
    $currPath \leftarrow nTuple(k)$
    for $j \leftarrow 1$ $to$ $k - 1$ do
        $validSubsets \leftarrow contains(S^j, currPath)$   ▷ Limit the available subsets to supersets of the current path
        $currPath_j \leftarrow drawRandom(validSubsets)$       ▷ Get a random subset
    end for
    $currPath_k \leftarrow i$                     ▷ The same path may get drawn multiple times,
                                                  ▷ but it still has to be in the set
    $\tilde{S} \leftarrow \tilde{S} \cup currPath$
end for
```

Algorithm 3: Pseudocode for path-superset sub-sampling

- **Path grouping**: For path or path-superset sub-sampling, m equals the number of paths $|\tilde{S}$ selected. Each Y_m then contains $k-1$ estimates: $Y_m = \{(\widehat{acc}_c(\tilde{S}_{m,i}, \tilde{T}_{m,i}), m) : i \in \{1, ..., k-1\}\}$. This results in multiple functions f^m.

3.2 Combining sub-estimates with curve fitting

3.2.1 Function models and fitting algorithms

The function model used has to be capable of modeling the learning process and also largely determines the spectrum of algorithms available. For a linear model or one that can be linearized, e.g. the 2-parameter exponential law, the computation of the parameters which minimize the squared error is well known and trivial. More complex algorithms are necessary for functions which cannot be linearized, like the 3-parameter exponential law. Then, iterative methods have to be used, like the Levenberg-Marquardt algorithm [28]. It works by iteratively adapting the parameters following its approximated gradient, with the goal to find a minimum for the least squares error function.

While they are able to handle a larger number of functions, they also need to be provided with various tuning parameters. In the case of Levenberg-Marquardt, initial values and maximal change per parameter as well as the partial derivatives w.r.t. the parameters must be given. Also, convergence is not guaranteed; unlike linear least squares, where minimizing parameters exist for at least two data points with different x components, poorly chosen initial parameters or too few iterations may lead to divergence. Potentially even more detrimental are local minima, as a divergence may be recognized. Here, the derivative of the error function is zero, indicating a minima, but different ones with lower absolute error values exist. However, the algorithm has no way of detecting this; the only options to avoid such a, quite literal, pitfall are to try the fitting with multiple initial parameters, hoping to get lucky, or to exploit previous knowledge about the data.

Potential function models were examined in section Chapter 2, especially in [38]. A good candidate for least squares fitting seems to be the 3-parameter exponential law in Equation 3.1.

$$f_E(x) = a + b \cdot e^{c \cdot x} \tag{3.1}$$

Unfortunately, it falls in the category ”non-linearizable” and requires iterative fitting. A similar function class are sigmoids. The evaluation in Chapter 4 investigates the characteristics of the sigmoid in Equation 3.2 in comparison to Equation 3.1.

$$f(x)_S = y_0 + 2 \cdot (y_0 - S) \cdot \left(\frac{1}{1 + e^{m \cdot x}} - 0.5 \right) \tag{3.2}$$

While it was not part of the testing in the cited articles, it can be similar in shape thanks to the exponential part and has the advantage of semantic parameters, that means they communicate the function's shape without the need to draw it. In this case, y_0 indicates the y-intercept, S is the asymptotic threshold, and m communicates the function's slope. This way, it is easier to find appropriate bounds for the parameters during fitting: clearly, a learning curve has to have both y-intercept and asymptote between 0 and 1 as well as a slope larger or equal to 0. While similar parameters can be found for the exponential function, they are not as precise, leaving more room for potentially wrong guessing, especially for the initial parameters.

As overfitting is a real possibility for all kinds of learning [13], the third function tested is the simple linear model of form

$$f(x)_L = a + b \cdot x \tag{3.3}$$

Since it does not approximate a whole learning curve of bent shape as seen in Figure 2.3, it will only be fit on accuracy estimates for subsets of size $j \in \{k-5, ..., k-1\}$, simply discarding the others. Thus, the linear fit approximates the momentary gradient of the classifier's accuracy rather than the whole process. Figure 3.2 exemplifies the general shape of all three functions when fit on data.

The final performance estimate for our classifier c_X is then obtained by extrapolating the functions f^m, i.e. $\widehat{acc}_c(X, \mathcal{D}) = \frac{1}{|f|} \sum_{i=1}^{m} f^m(k)$. In case of $|f| > 1$ the $f^m(k)$ can be used as samples of a probability distribution, opening up the possibility of comparing not only the difference of $\widehat{acc}_c(X, \mathcal{D})$ to $acc_c(X, \mathcal{D})$, but also how different the (estimated) distributions of acc_c over $(\mathcal{D})$ are, which is explained in detail in Section 4.1 as part of the measurements used to compare the different estimators.

3.2.2 Tuning the model fitting

Usually, data gathered for curve fitting is not uniform, e.g. some data points are more likely to be tainted with error or do not carry much information. An example could be

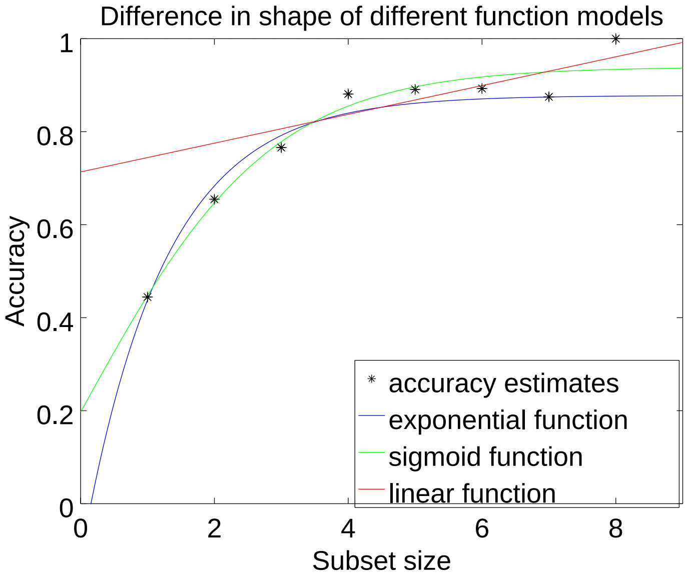

Figure 3.2: The shape of the three function models fit on example estimates produced by *averaged grouping* with $k = 9$ for the *seeds* dataset

a study questioning people about the value of the elementary electric charge. With a sufficiently large sample size, averaging the individual statements may produce a fairly rough approximation to the real value. However, if the answers of people related to physics (students, engineers etc.) would be counted twice, the approximation would likely be better, as these people generally know more about physics and are thus more likely to know the correct answer.

A similar situation presents itself when using averaged grouping, where the subset estimates $\widehat{acc}_c(S_i^j, T_i^j)$ for each j are first averaged, then sent to the fitting algorithm. However, this procedure omits how many estimates were originally present, which is quite important information; more estimates mean the resulting average should be less noisy. This is where statistical weights come into play: they inform the fitting algorithm about how important a low error of the fitted function for a given data point is, the difference in shape for an example exponential function is depicted in Figure 3.3. Assigning weights proportional to the amount of estimates that went into each $Y_{m,i}$ may ensure that less noisy data influences the shape of the curve more heavily.

Another potential improvement for the fitting process is the addition of data. Of course the number of subsets is fixed without purchasing more labels. But information not covered by leave-p-out may prove useful: showcased in [15], .632+ bootstrapping uses the *no-information rate* to identify special overfit cases with data independent of its predictors. It is a heuristic used to approximate the error a classifier without any training would make when being tested on the dataset, the formula can be found in Equation 2.19. And while performance estimates for each subset size $|S_i^j| \in \{1, ..., k-1\}$

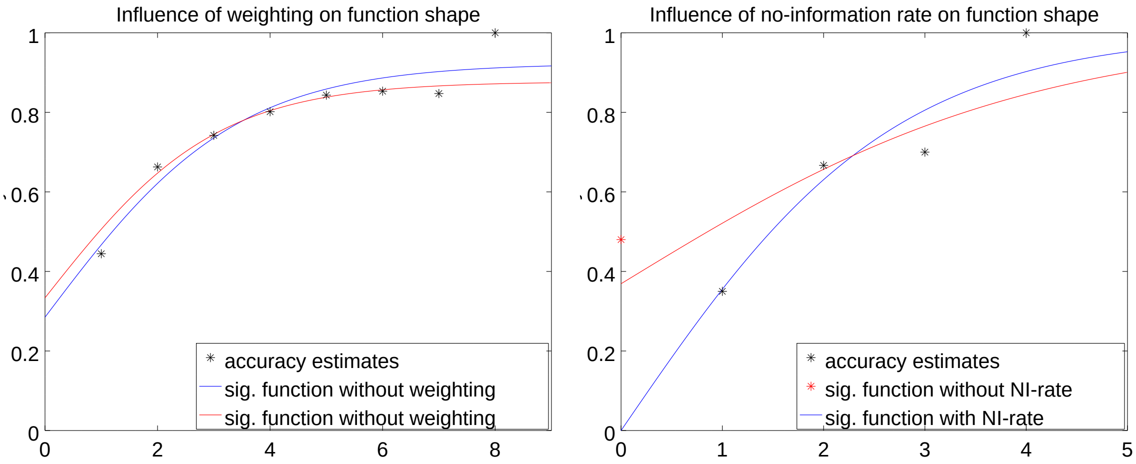

Figure 3.3: Difference of function shape for the sigmoid model when using weighting or the no-information rate

are available, the estimates do not yet include an estimate for $j = 0$, since most classifiers require at least one training instance to work. The no-information rate may be used as the 0th fitting point as seen in Figure 3.3, filling that gap and help primarily in scenarios with small k. For larger training sets, its influence should decrease as more regular estimates are available.

4. Evaluation

This chapter contains the discussion of the evaluation methodology, which comprises of measures to determine how well different estimators perform, as well as the structure and results of our testing.

4.1 Performance criteria

The selection of measurements for comparison should be guided by what is expected of a performance estimator. Highest on the priority list is the requirement that its estimate $\widehat{acc_c}$ is equal or at least close to the true value acc_c. This criterion is actually bipartite: how large is the difference between the two on average and does it show a preference. The latter is commonly referred to as *bias*, while in the following, the former will be called *spread*. When comparing two estimators, the primary focus is on the bias; a low spread is not exactly useful when it is centered around a wrong value.

The estimation of bias and spread are done, based on measures used in [17], by computing the *mean error (ME)* and the *mean squared error (MSE)*:

$$ME(k) = \frac{1}{n} \sum_{X \in \mathcal{X}^k} \left(acc_c(X, \mathcal{D}) - \widehat{acc}_c(X, \mathcal{D})\right)^2 \tag{4.1a}$$

$$MSE(k) = \frac{1}{n} \sum_{X \in \mathcal{X}^k} \left(acc_c(X, \mathcal{D}) - \widehat{acc}_c(X, \mathcal{D})\right)^2 \tag{4.1b}$$

where $\mathcal{X}^k$ contains the different training sets X used in testing of size k. Lower values for ME and MSE indicate a better estimator.

Depending on the method in question, additional measures for comparison may be available. As a side effect of the multiple functions fit when using path or path-superset sub-sampling, the resulting estimates can be seen as sample points. The mean and variance of them specify a probability distribution. Luckily, the choice of the distribution model to assume is fairly simple, as most models are not suitable anyway since they are defined on $\mathbb{R}$. Our estimates, however, are limited to $[0, 1]$. Thus, only few distributions come to mind, one of which is the beta distribution, also being used for similar purposes by [25]. It is dependent by two parameters, $a, b \in \mathbb{R}_{\leq 0}$, with a density function defined as [18]

$$q_{a,b}(x) = \frac{x^{a-1}(1-x)^{b-1}}{\int_0^1 u^{a-1}(1-u)^{b-1}du}. \tag{4.2}$$

The parameters are computable from mean and variance via

$$\begin{aligned} E[Q] = \mu = \frac{a}{a+b} &= \frac{1}{|\mathcal{X}^k|}\sum_{X\in\mathcal{X}^k}\widehat{acc}_c(X,\mathcal{D}) \\ var[Q] = s^2 = \frac{ab}{(a+b)^2(a+b+1)} &= \frac{\sum_{X\in\mathcal{X}^k}(\mu - \widehat{acc}_c(X,\mathcal{D}))^2}{|\mathcal{X}^k|-1} \end{aligned} \tag{4.3}$$

Solving for a and b results in

$$\begin{aligned} a &= \frac{\mu^2(1-\mu)}{s^2} - \mu \\ b &= a\left(\frac{1}{\mu} - 1\right) \end{aligned} \tag{4.4}$$

The function q is then the *probability distribution function (PDF)* of the estimated accuracy distribution of $\widehat{acc}_c$. Now, the measure to compare the estimated and the *true* accuracy distribution denoted by P is the *Kullback-Leibler divergence (KLD)*:

$$KLD(P:Q) = \int p(x)\cdot log_2\left(\frac{p(x)}{q(x)}\right)d\lambda(x), \tag{4.5}$$

$p(x)$ is the PDF of the distribution P [27]. The KLD is an information-theoretical measure intended to compare two probability distributions. If the two are equal almost everywhere, it equals zero, otherwise returns a positive value. Importantly, it is neither symmetrical nor does it satisfy the triangular inequality, thus the choice of distribution assignment is meaningful and should be equal for all tested methods (i.e. the distributions Q must not be swapped) [20]. As the integral cannot be evaluated in closed form, numerical integration is used instead:

$$KLD(P:Q) = \frac{1}{|N|}\sum_{n\in N} p(n)\cdot log_2\left(\frac{p(n)}{q(n)}\right) \tag{4.6}$$

with $N = \{\frac{1}{10000}, ..., \frac{9999}{10000}$. Due to the lack of floating point precision in Octave, which was used to implement the framework, the edge cases 0 and 1 were omitted; the values there for p and q were mostly zero or NaN.

Another important aspect of an estimator is its time complexity: only marginal improvements in bias of an estimator compared to its peers are unlikely to compensate for a drastic increase in computation time. Thus, it is also regarded in Section 4.4.

4.2 Method selection

The method described in section Chapter 3 is build as a three-step process: first, some sort of sub-sampling for the subsets is performed (whether this actually reduces the amount of subsets is irrelevant). Then, the performance for the selected subsets is estimated and grouped. Last, the function model of choice is used to fit a curve on the groups and the final performance estimate obtained by extrapolating the functions to the set size k and averaging the results. Including the possibility of using fitting improvements like weighting and the no-information rate as well as different function models, a lot of modules for each stage are available. Unfortunately, I cannot test all combinations, as it is quite time consuming.

Of course, some of the stage modules are incompatible: when using capped sub-sampling, applying path grouping is not possible, as there are no paths. Likewise, when path sub-sampling has been chosen, only path grouping is viable. Also, pre-screening suggested that the usage of the no-information rate as zero-set-size estimate is not as effective as hoped. Causes seem to be both high variance, leading to worsened fitting with low amounts of sub-samples, as well as its expendability for larger sub-sample sizes, which grow with the size of the training set.

Method name	Description
5-Fold CV	5-Fold cross-validation
.632+ BS	.632+ bootstrapping with 50 bootstrap samples
path	Path sub-sampling with path grouping
pathSuper	Path-superset sub-sampling with path grouping
averaged	Capped sub-sampling with averaged grouping; cross-validation for estimation
averagedBS	Capped sub-sampling with averaged grouping; .632 bootstrap for estimation
pathW	path with statistical weighting for fitting
pathSuperW	pathSuper with statistical weighting for fitting
averagedW	averaged with statistical weighting for fitting
averagedBSW	averagedBS with statistical weighting for fitting
pathSuperNI	pathSuper with the no-information rate as estimate for subset size 0

Table 4.1: Summary of all estimators evaluated in the tests

As a result of this, the testing is conducted for 11 different estimators: path and path-superset sub-sampling combined with path grouping, which will be abbreviated as **path** and **pathSuper**, as well as capped sub-sampling with averaged grouping with the name **averaged**. While the estimation technique for the two path-based methods will be exclusively cross-validation, *averaging* is also tested with .632 bootstrapping with the identifier **averagedBS** to assess whether it has any impact on the quality of the methods. Additionally, each of these estimators are evaluated using both weighted and unweighted fitting, with the former being designated by a trailing *W*. Additionally, to confirm the suspicions raised in our pre-screening, *pathSuper* is also tested in a modified version with a 0th data point for each path/curve, named **pathSuperNI**. To complement the selection, the more traditional estimators 5-fold cross-validation and .632+ bootstrapping also participate as **5-fold CV** and **.632+ BS**. Table 4.1 lists all participating estimators and their description.

Additionally, each of these estimators is tested with the three potential function models explored in Section 3.2.1: exponential, sigmoid, and linear model. Each graph's designation contains either *"exp."*, *"sig."* or *"lin."* to indicate the model used.

4.3 Test environment

4.3.1 Reference accuracy

The ground truth $acc_c(X, \mathcal{D})$ is extracted with holdout testing. For this, all instances in $\mathcal{L}$ which are not part of X are split in m sets H_i of size k; for the evaluation, $\mathcal{L} = \mathcal{D}$ since all instances are accompanied by a class label in the datasets used. Then, the accuracy of the classifier x_X on each H_i is computed. The holdout accuracy acc_c then equals the average accuracy over all H_i: $acc_c(X, \mathcal{D}) = \frac{1}{m}\sum_{i=1}^{m} acc_c(X, H_i)$. The $acc_c(X, H_i)$ also are used as the sample points from which the true accuracy distribution P is computed: $\mu = acc_c(X, \mathcal{D})$, $s^2 = \frac{1}{m-1}\sum_{i=1}^{m}(\mu - acc_c(X, H_i))^2$.

4.3.2 Function fitting

As neither the exponential nor the sigmoid function model are linearizable, the fitting used the Levenberg-Marquardt algorithm, which, amongst others, requires the specification of initial parameters. Also, as a preconceived image of a learning curve as monotone rising and limited to the interval $[0, 1]$ exists, providing constraints to fulfill these requirements is reasonable. This is trivial for the sigmoid function, as it was specifically designed to allow this kind of modification: y_0 and S, as representations of the y-intercept and the function's asymptote, are limited to $[0, 1]$ with $y_0 \leq S$, and m has to be in $\mathbb{R}_{\geq 0}$. For the other functions it is not as easy to find suitable bounds: while both have parameters directly affecting the y-intercept and slope, their asymptote for $x \mapsto \inf$ is either affected by two parameters or is not a constant (linear). Both would require polynomial equality constraints, which were not taken into account for this work. Thus, it is good to keep in mind that these functions might violate the learning curve constraints.

4.3.3 Active learner

All tests are conducted with three different active learners: random sampling, uncertainty sampling and PAL; which one is indicated in the graphics. For their functioning see Section 2.2. Although it does not matter due to the dichotomous nature of the used datasets, uncertainty sampling uses the maximum entropy for instance selection.

4.3.4 Classifier

The choice of an adequate classifier is mostly limited by the active learners and the datasets used in the evaluation. From the dataset side, it has to be able to accept continuous features. The output of class assignment probability is necessitated by PAL and uncertainty sampling. Also, PAL requires some sort of density estimation; to keep the comparability of both active learners up, a classifier making use of kernel density estimation (KDE) is reasonable. Considering this, Parzen window lends itself to be the classifier of choice. It is a non-parametric classifier directly build on KDE. The assumption is that the data is distributed according to some probability distribution; basic assumption is the normal distribution. Then, the kernel K, which in our case is the probability density function (but it does not have to be), is approximated at a given instance $\vec{x}$ using the following formula:

$$\hat{f}_h(\vec{x}, X) = \frac{1}{h \cdot |X|} \sum_{\vec{x'} \in X} K(\frac{\vec{x} - \vec{x'}}{h}) \tag{4.7}$$

As the training set, X contains the already known instances and h is the so-called bandwidth, a smoothing factor for the kernel [36]. It has to be estimated, which is usually done by applying a function called *Silverman's rule of thumb*, which is dependent on n, the dimensionality of the data, its estimated standard deviation and some statistical properties of the kernel largely irrelevant to this work. The classifier used in the evaluation however simply assumed a standard deviation of 0.1 across all dimensions. If the data in question is multivariate, Equation 4.7 has to be a bit modified; K is then usually the product of the kernel for each dimension, same goes for the bandwidth [37].

The Parzen window classifier estimates the densities of an instance to be labeled for each class label, each time using only the instances with the same label as $\vec{x}$. Then, they are multiplied with the prior class probabilities, i.e. the share each class label has among the labeled instances. Normalization then results in the wanted (estimated) class probabilities, with the largest probability dictating the resulting class [3]. Figure 4.1 shows the classification of a Parzen window classifier: the color expresses the predicted class label, while the z coordinate equals the kernel density with either one positive or two negative instances as its base, depending on which one is larger.

4.3.5 Datasets

The datasets used should both be realistic and cover most uses; a method that does well on specifically designed test sets but fails in the real world is only interesting as a proof-of-concept. Secondly, PAL was only defined for dichotomous data. Although an extension on multiple-class problems should be possible, I did not want to tamper with the formulas, instead restricting the datasets to binary-class problems.

Considering these constraints the selection contains the following datasets:

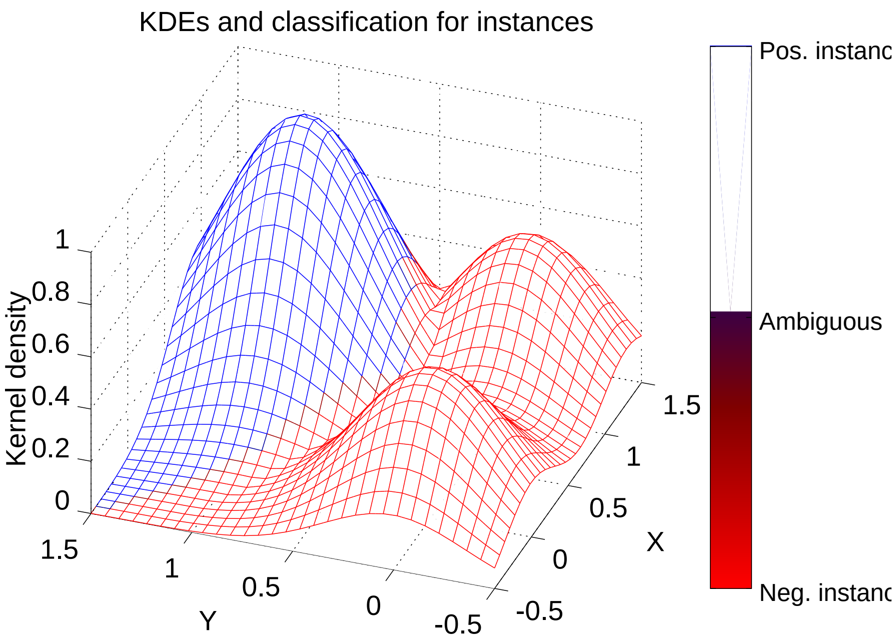

Figure 4.1: The estimated kernel density for a grid with one positive and two negative instances; lower Z value indicates lower certainty for the class assignment

- **checke1**: This artificial dataset was used in [8] to examine active learning with Parzen window classifiers and contains 400 instances with two features. It has the form of a 4x4 checkerboard, with only every second field containing instances. There is no overlapping of class labels, i.e. the instances can be perfectly separated by class label using decision boundaries. This set may be problematic for uncertainty sampling as it will select instances at its decision boundary, while the set requires *multiple* decision boundaries. Thus, it likely will not label instances from the other fields until it runs out of close ones. Both random sampling and PAL should not have this problem; the former does not care about any structure anyway, while the latter actively explores the ”uncharted” instances.

- **2dData**: Also an artificial, 2-dimensional dataset, it was used in a follow-up paper on optimized PAL [24] under the name ”Sim”. The 1200 instances are grouped into two clusters of roughly ellipsoid shape, but cannot be perfectly separated as they slightly overlap, making it seem more realistic (real-world datasets tend to have some noisy data). None of the active learners should have inherent problems with this set.

- **seeds**: A real-world dataset used in [9]. It has seven features which describe different properties of wheat, classifying it into three varieties: Rosa, Kama and Canadian with 70 instances each. To comply with the constraints set earlier, the varieties Kama and Canadian were merged into one class. As it is 7-dimensional, obtaining a visual is difficult; thus ”t-Distributed Stochastic Neighbor Embedding (t-SNE)”, a method for dimensionality reduction of high-dimensional data [39] to project it into $\mathbb{R}^2$ by using Gauss kernels to keep neighboring instances together, rendered the visualization. Similar to *2dData*, two clusters seem to be present, each containing instances sharing the same class label, albeit a small overlap exists. Due to the similarity, no problems regarding the active learners are to be expected.

- **abalone**: A real-world dataset using eight features associated with predict the age of abalones. Originally, the number of rings (and thus ages) of a specimen were predicted. To create a binary set, all ages below 10 form one class, while the rest forms the other. The set was obtained in the study [29]. As the original set contained 4177 instances, it had to be reduced to remain a viable option, considering the finite amount of time available for testing. Thus, the number of instances was reduced to 1800, keeping the class ratio intact. The visualization indicates the presence of four separate clusters with partially mixed class labels. While this may hint at problems with uncertainty sampling, the study itself suggests that the predictors are not sufficient for classification, leading to high error rates even with large training sets.

4.3.6 Parameters

Some of the methods as well as the fitting requires the specification of parameters like the number of iterations or error tolerance. As their choice may influence the results, all parameters used will be stated in this section.

For each combination of dataset, active learner and function model, 100 runs were simulated, each purchasing one instance at a time up to 30 and making estimates each step. All function models were fit on the same data.

For the *path* and *pathSuper* estimators, k^2 but not more than $\frac{10000}{k}$ paths were randomly selected. *Averaged* makes use of all subsets up to 10000 (due to rounding may not be exact); its bootstrap variant *averagedBS* is allowed up to 50 subsets as well as 50 bootstrap samples per subset. Both caps were placed due to memory and time limitations. The modified variants with weighting/no-information rate use the same subsets/paths for any given test run. *5-fold CV* has no limitations in place; *.632 BS*, similarly to *averagedBS*, also uses 50 bootstrap samples per estimation.

All functions were fit with the Levenberg-Marquardt algorithm, even the linear model; this was done to normalize the testing environment. The ranges for initial parameters as well as their bounds can be seen in Table 4.2. Each fitting can use up to 300 iterations with a scalar tolerance of 10^{-4}, i.e. if the sum of squared errors is lower than 0.0001, the fitting is considered done and will not use the remaining iterations. The partial derivatives were provided, thus numeric derivation was not necessary.

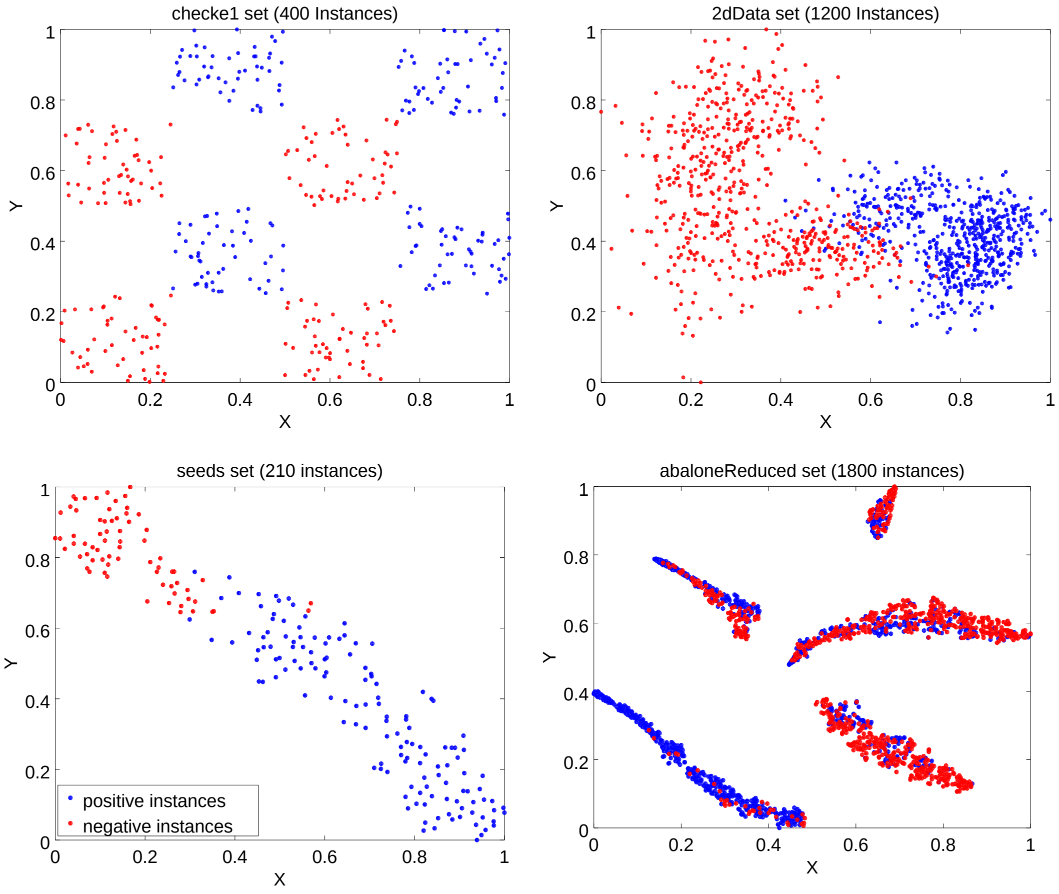

Figure 4.2: Visualizations of the datasets checke1, 2dData, seeds and a downsized version of abalone [8, 9, 25, 29].
The illustration of seeds and abalone was done using an implementation of t-SNE [39]

As Levenberg-Marquardt may be stuck in local minima or not converge at all, each fitting was done 5 times with random parameters drawn from their respective ranges to attempt to circumvent these hazards. Then, the fit with the lowest sum of squared errors w.r.t. the data given was selected. If the fitting used statistical weights, they were also used in this selection.

4.4 Test results

Due to the sheer amount of data, this section contains only the most expressive graphs. To help keeping an overview, we structure the evaluation into various parts: first, we take a look at the mean error for the traditional and unweighted methods using the exponential function model. After that, we evaluate if using a different function model for fitting improves the bias as well as the influence of statistical weighting. Then we look at the average squared error and the Kullback-Leibler divergence where possible. Lastly, an analysis of the computation time necessary for each method follows.

Function model	Equation	Parameter bounds	Initial parameters
3-Exponential	$f_E(x) = a + b \cdot e^{c \cdot x}$	$a \in [0, 1]$ $b \in [-\infty, 0]$ $c \in [-\infty, 0]$	$a \in [0, 1]$ $b \in [-2, 0]$ $c \in [-2, 0]$
Sigmoid	$f_S(x) = a + b \cdot e^{c \cdot x}$	$a \in [0, 1]$ $b \in [-\infty, 0]$ $c \in [-\infty, 0]$	$a \in [0, 1]$ $b \in [-2, 0]$ $c \in [-2, 0]$
Linear	$f_L(x) = a + b \cdot x$	$a \in [0, 1]$ $b \in [0, \infty]$	$a \in [0, 1]$ $b \in [0, 4]$

Table 4.2: Function-specific parameters for model fitting

4.4.1 Average Mean Error

To inspect whether our estimators carry a bias and if yes, how large it is, we compute the mean error between the holdout accuracy and that of our methods for each training set size and test run. Since some stages of the learning process may be easier for the estimators than others we group the error by the set size they were measured for. Figure 4.3 shows the average mean error for the training set sizes from $3-7$, $8-15$ and $16-30$, marked by decreasing color brightness.

Generally, all performance estimators show less bias for random sampling, with the worst bias for uncertainty sampling except for the checke1 dataset. Overall, the least biased estimator seems to be *.632+ BS*, with *k-fold CV* close second. The only time they show a higher bias than our methods for random sampling is on the abalone dataset, which is not to be attributed to a failure of them as much as to a better performance of the combined estimators. The .632+ estimator also seems to have trouble when used with PAL, leaving it with a higher bias for both checke1 and abalone than both k-fold and our methods. It is to be kept in mind that, although they are displayed as a stacked bar, the mean errors for the different learning stages do not accumulate, e.g. the average mean error for averagedBS with uncertainty sampling on 2dData is not 0.8 for the latest stage but rather 0.35.

Interestingly, most of the time the estimators predict a lower classifier performance than it actually has. Only *.632+ BS* seems to show no preference with random sampling, hinting at a reasonable unbiasedness. For uncertainty sampling, however, all estimators underestimate the performance, at least for the later stages. When using PAL, no clear pattern for the bias sign can be made out, but the estimators mostly share the sign for the same dataset.

Looking at the average mean error with the sigmoid function model in Figure 4.4, we can see that it does offer improvements over its exponential counterpart. While the general trends regarding the bias sign and relations between the active learners is roughly the same, our methods seem to benefit greatly from using the sigmoid model. All of them show lower biases for random sampling, now roughly equal to that of *k-fold CV* but still worse than *.632+ BS*. This time, a hierarchy seems to manifest, with *averaged* exhibiting the least bias and *averagedBS* marking the upper end. A slightly

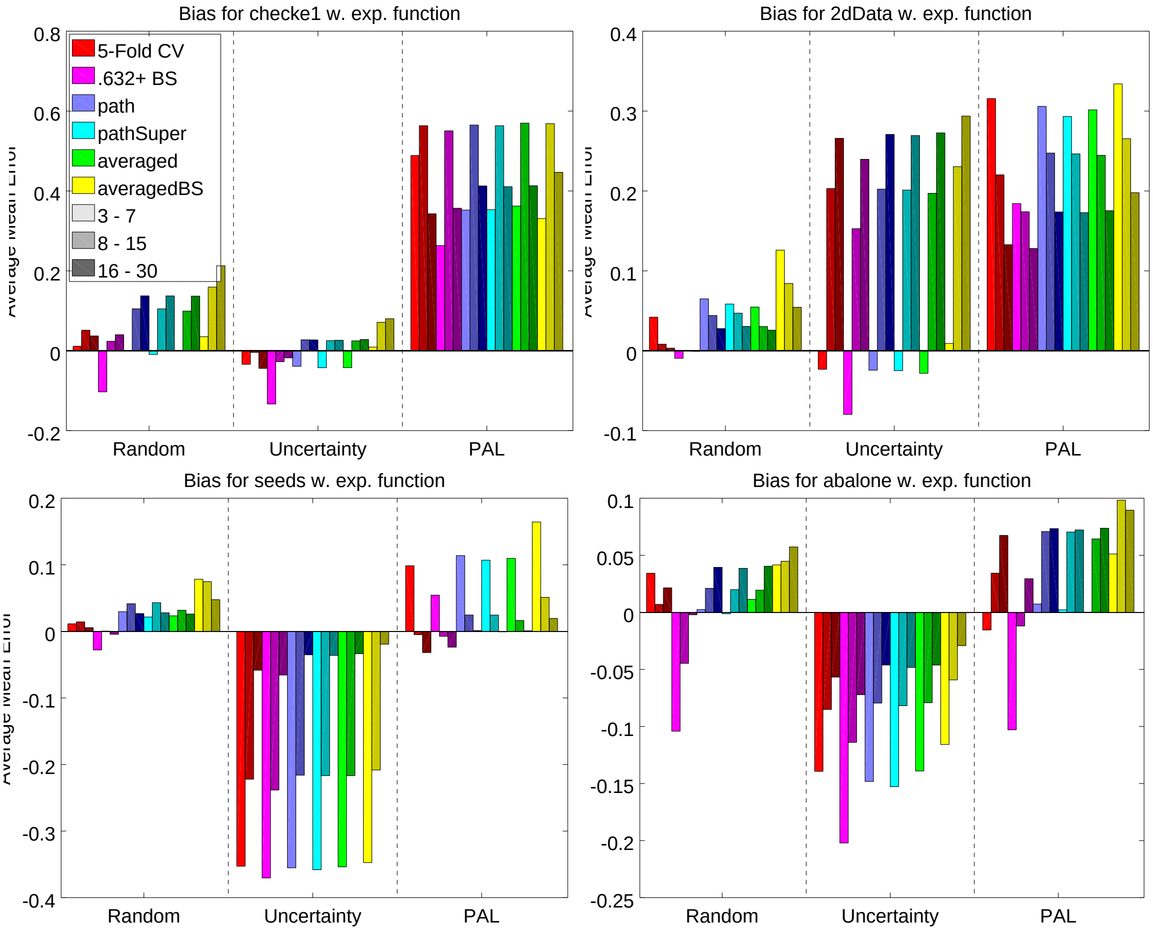

Figure 4.3: Average mean errors for the different active learners and datasets using the exponential model. The darker colors of a bar mark the errors of later learning stages (bright -> dark: $[3, 7]$, $[8, 15]$, $[16, 30]$ training set size)

different picture is painted when using non-random active learning: although the bias rarely deteriorates, neither does it improve significantly, mostly staying on the same level. Also, no real hierarchy seems to be present; which method does best depends on the dataset and the active learner, although *averagedBS* is mostly still the tail light.

Very differently from its two brothers is the linear model. As depicted in Figure 4.5, most of our estimators *over*estimate the classifier's performance, except for the use-case of uncertainty sampling. For random sampling, *averagedBS* is actually an improvement over *k-fold CV* and seems to be roughly on the same level as *.632+ BS*. *pathSuper* is close behind, but in most cases a bit worse, while the "normal" *averaged* actually deteriorated in comparison to the exponential and sigmoid functions. Similar statements are true when using PAL, but it seems a bit more random; on the seeds dataset, *averagedBS* shows a higher bias than the other estimators, while on the rest of the datasets it even outperforms *.632+ BS*. This hierarchy is reversed for uncertainty sampling, with *averaged* apparently being the least biased.

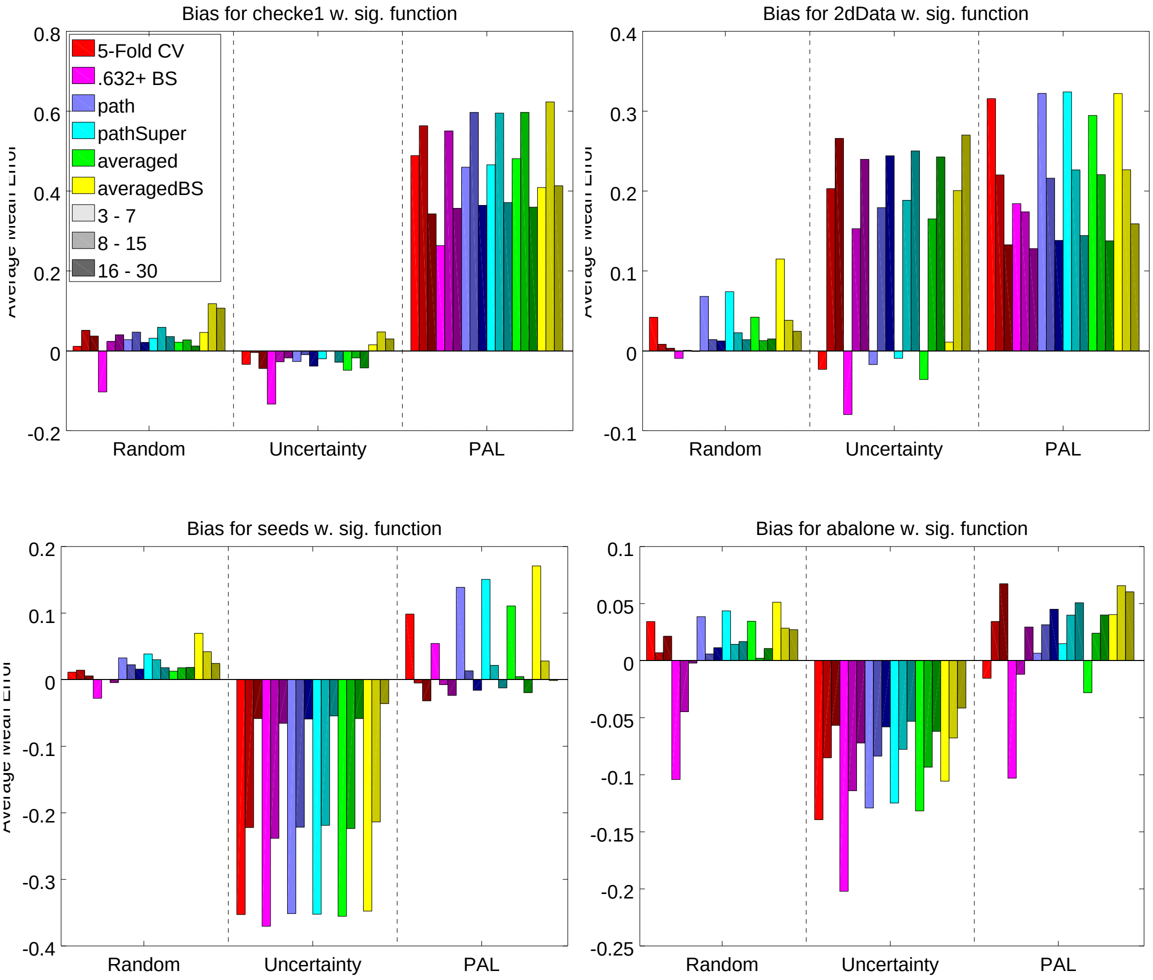

Figure 4.4: Average mean errors for the different active learners and datasets using the sigmoid model. The darker colors of a bar mark the errors of later learning stages (bright -> dark: [3, 7], [8, 15], [16, 30] training set size)

An interesting phenomenon shows when using uncertainty sampling. While the bias for the first seven training set sizes usually makes up more than 30% for the other two active learners, its share otherwise is incredibly small. In other words, the bias increases with training set size. We suspect that this is caused by the way the next instance is selected: since the class probabilities predicted by the classifier will be close to 0.5 each, we will get an error near 0.5 on average with cross-validation. As all of the estimators are based on the principle of leaving instances out of the training set, they all carry this flaw.

In summary, the best estimators (besides the traditional ones) seem to be *averaged* with the sigmoid model and *averagedBS* with linear fitting. To see what influence statistical weighting has on the bias, Figure 4.6 compares the average mean errors of

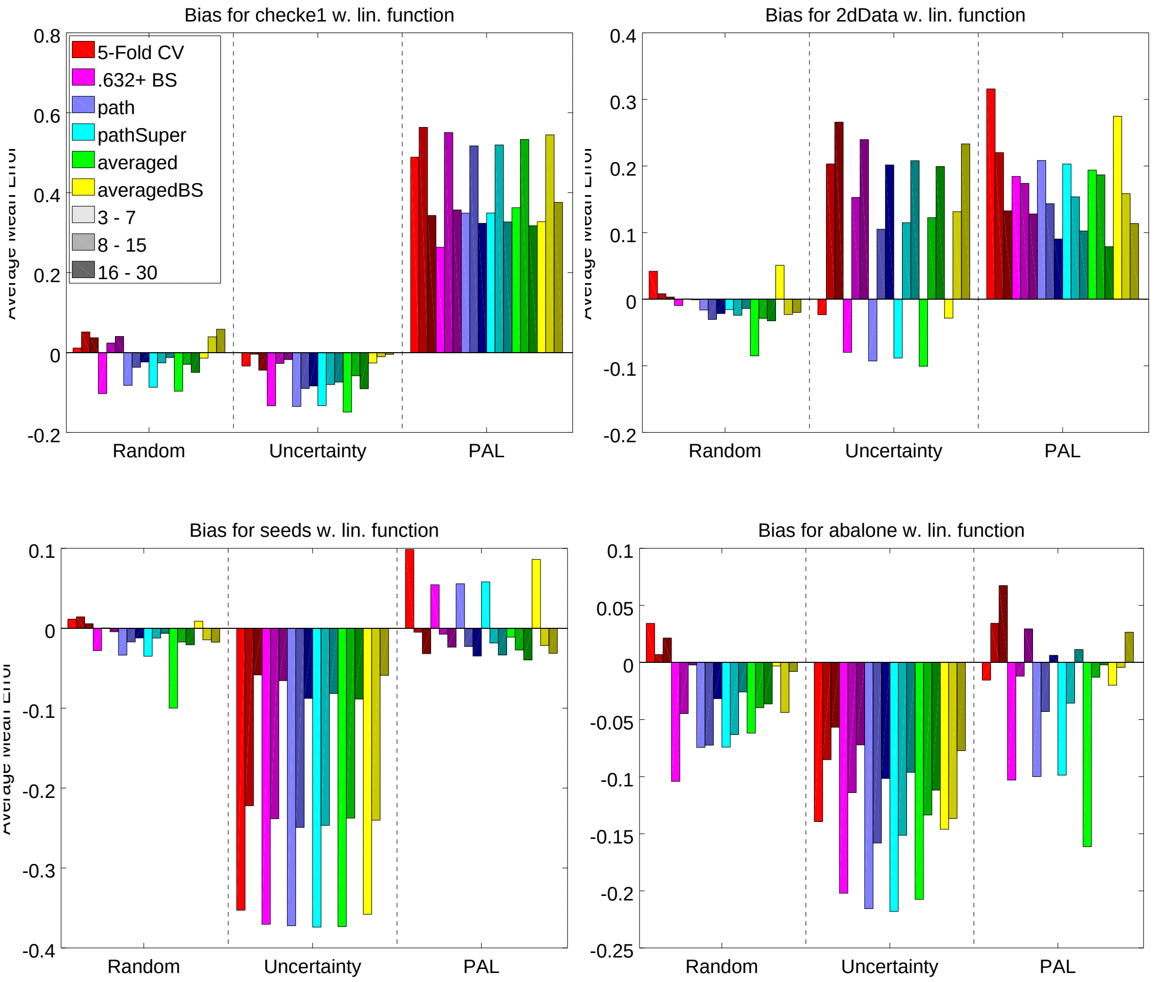

Figure 4.5: Average mean errors for the different active learners and datasets using the linear model. The darker colors of a bar mark the errors of later learning stages (bright -> dark: $[3, 7]$, $[8, 15]$, $[16, 30]$ training set size)

our estimators for random sampling. For the linear model, weighting mainly affects the early learning stages positively, with the rest stays more or less the same, while the bias gets worse when using the sigmoid model. The greatest improvement can be observed for the *pathSuper* method, where weighting consistently reduces the bias, in part even below that of *.632+ BS*. Contrary, the inclusion of the no-information rate has almost no influence whatsoever.

4.4.2 Average squared error

Next in line is the evaluation of the spread of error. As an overly biased estimator is of little use, we limit our testing for this to the six best ones: the two traditional estimators, *k-fold CV* and *.632+ BS*, *path* and *averaged* with the sigmoid model, as well as *pathSuperW* and *averagedBSW* with the linear model (Figure 4.7).

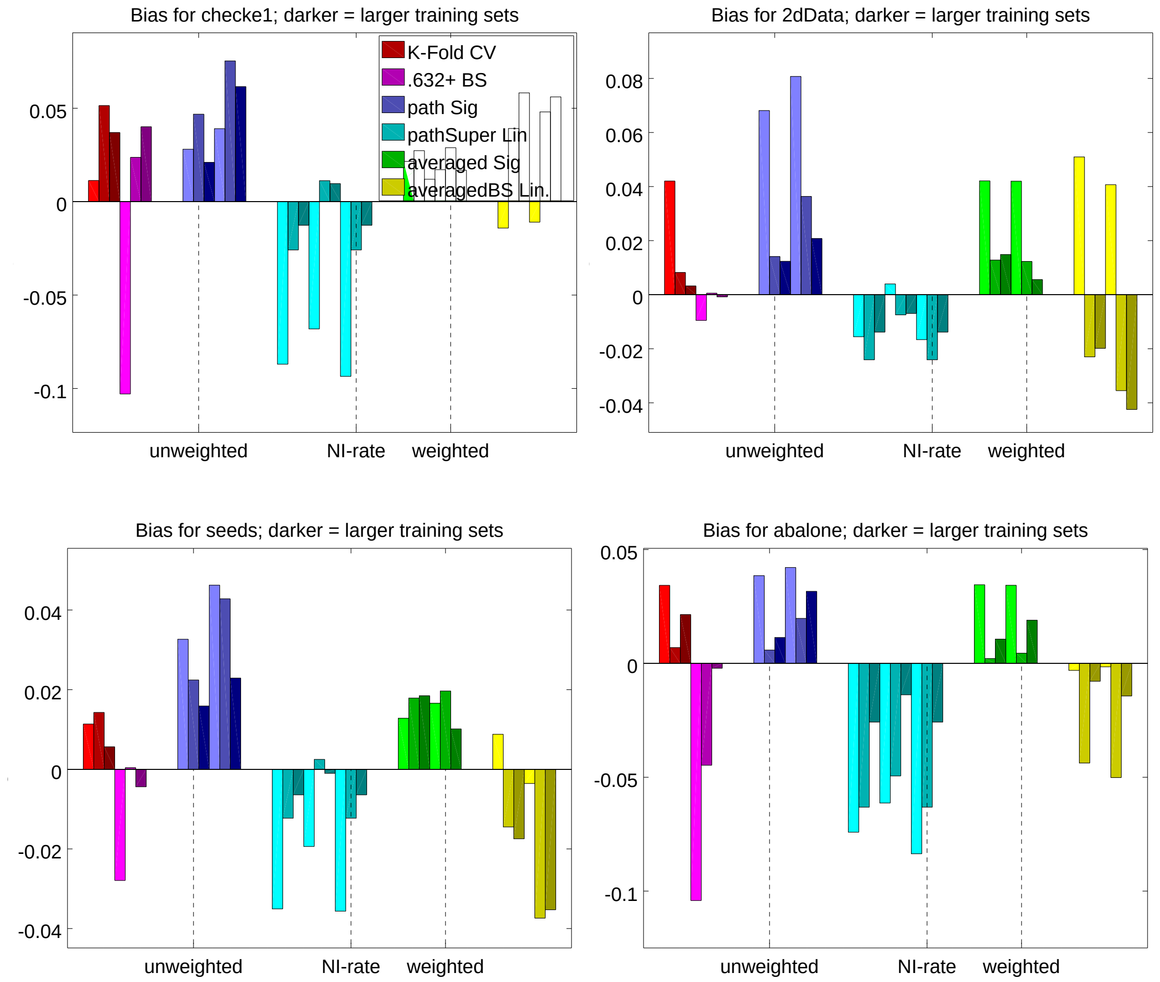

Figure 4.6: Average mean errors for different datasets with random sampling. The darker colors of a bar mark the errors of later learning stages (bright -> dark: $[3, 7]$, $[8, 15]$, $[16, 30]$ training set size)

Similar to the mean error, the squared error is generally higher for the non-random active learners, the only exception is for the abalone data. For the most part, *k-fold CV* and the two averaging methods have the highest spread while *.632+ BS* and *pathSuperW* mark the lower end. Something all estimators share is the reduction of spread for later learning stages; this seems natural, considering that more estimates are available then. As always, this is not the case for uncertainty sampling, where the spread, just like the bias, increases for the most part with training set size. Puzzling is also the increase and following drop-off for PAL on the checke1 set, a phenomenon that could already be observed when investigating the mean error. A possible explanation lies within both the structure of the dataset as well as the instance selection of PAL: checke1 consists of several separated groups of equally labeled instances. Figure 4.8 shows that, albeit

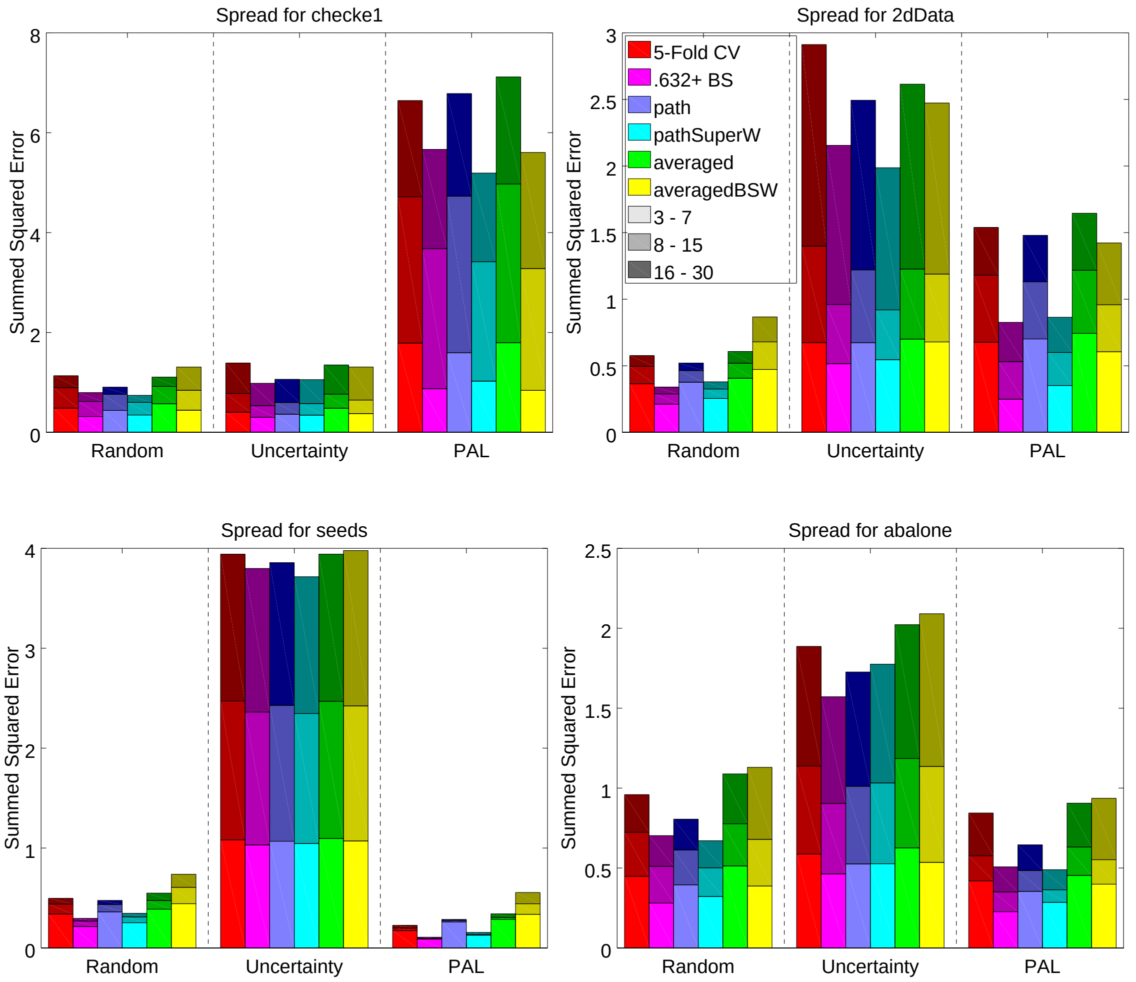

Figure 4.7: Average squared errors for the different active learners, function models and datasets

more consistent than uncertainty sampling, PAL takes some iterations to explore the whole dataset.

Noteworthy is the reduction of estimations and paths used from their possible upper limit, especially for *averagedBS* which only uses a maximum of 50 subset estimates. Increasing these caps would likely result in a reduction of the average squared error, but only for the later learning stages. However, this would also increase the (already quite high) computation time and should not reduce the bias, considering that the subsets were randomly selected with respect to their size frequency.

4.4.3 Kullback-Leibler divergence

The average Kullback-Leibler divergences shown in Figure 4.9 do not hold any real surprises. For random sampling, *k-fold CV* offers the smallest divergence, while both

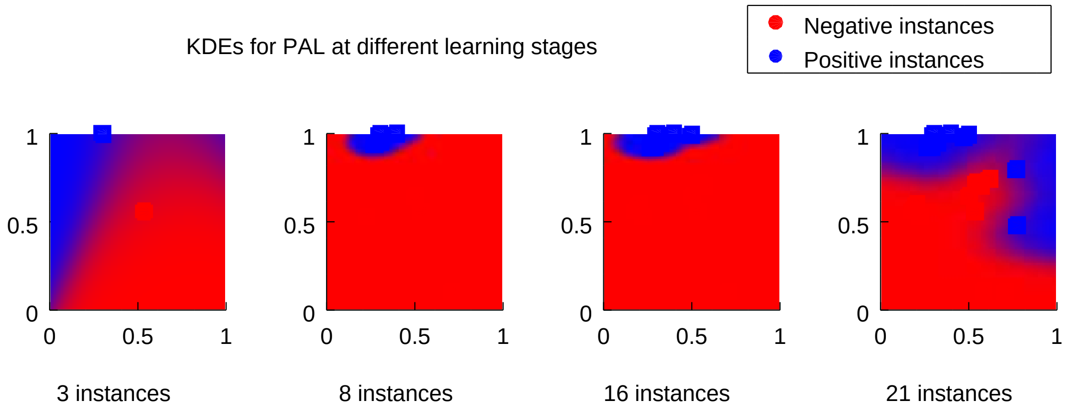

Figure 4.8: Kernel density estimations from the learning process of PAL for checke1 at 3, 8, 16 and 21 training instances

pathSuper estimators excel for the non-random learners. Neither of the *path* methods seem to be suitable to model the holdout accuracy distribution, as their divergences are quite high. Surprisingly, the average divergence rises across the board for larger training sets, which is quite in contrast to the average mean error. However, we do not read too much into this; after all, the estimators are supposed to estimate the value of the accuracy, not its underlying distribution.

4.4.4 Computation Time

Bringing the evaluation to an end, Figure 4.10 shows the average computation time needed for each method with the exponential function model. As statistical weights should not alter the time too much, we omitted these variants. It gets clear immediately that none of our methods are anywhere near as fast as either *k-fold CV* or *.632+ BS*. In fact, if all available estimates/paths would be used, all of them would scale exponentially with the training set size. Especially expensive is the bootstrap averaging, as bootstrapping itself is quite a bit more complex than cross-validation. It also has to be said that the choice of programming language may play a big role; Octave is not exactly known for its high performance and is even worse when not using properly vectorized data.

The computation time also seems to be independent of the used dataset, which is unexpected; the used classifier, parzen-window, scales with the dimension of the instances. Apparently this additional effort is negligible in relation to the cost of the fitting and subset generation and selection. Also pictured in Figure 4.10 is a histogram for the computation times for *pathSuper* using exponential fitting on checke1. Its spread is partly due to the variability in number of iterations for the fitting, but mostly because estimations for all training set sizes were used.

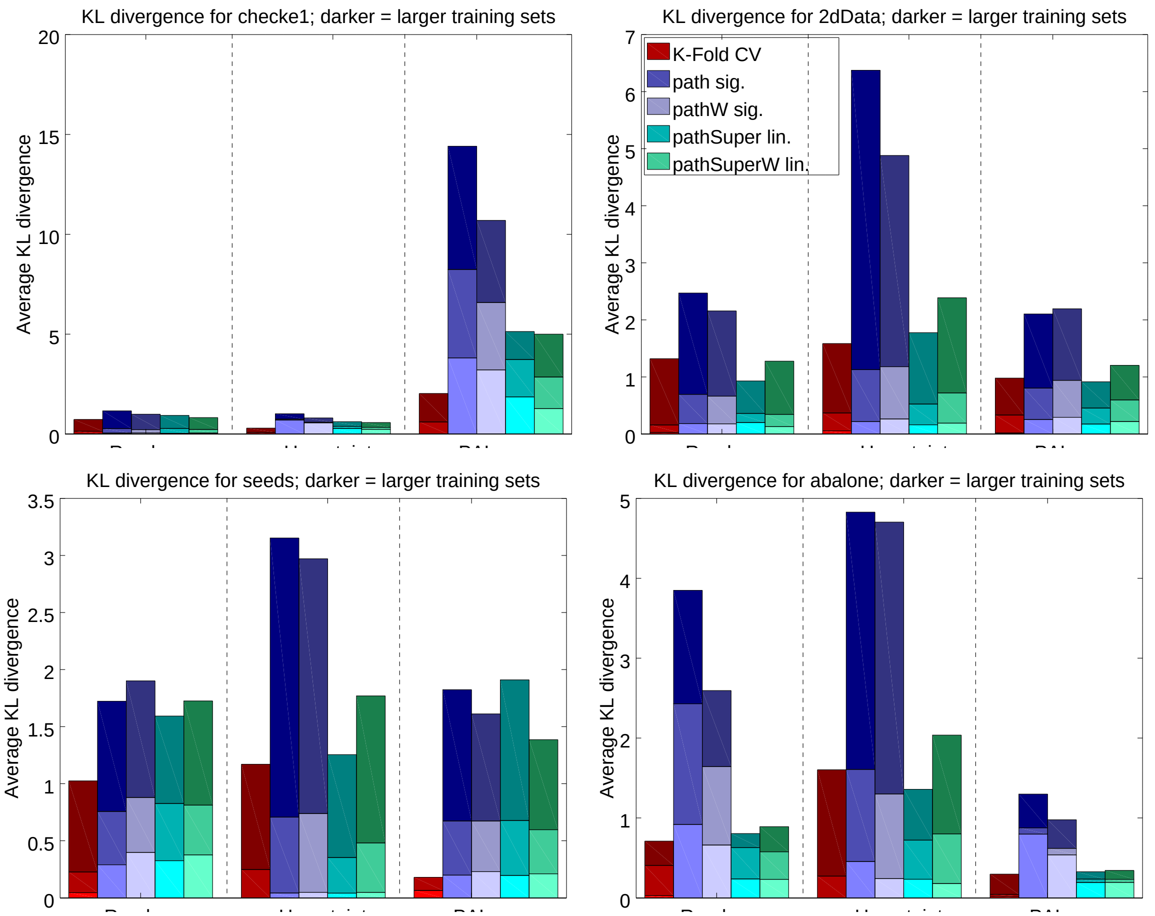

Figure 4.9: Average Kullback-Leibler divergence for selected methods

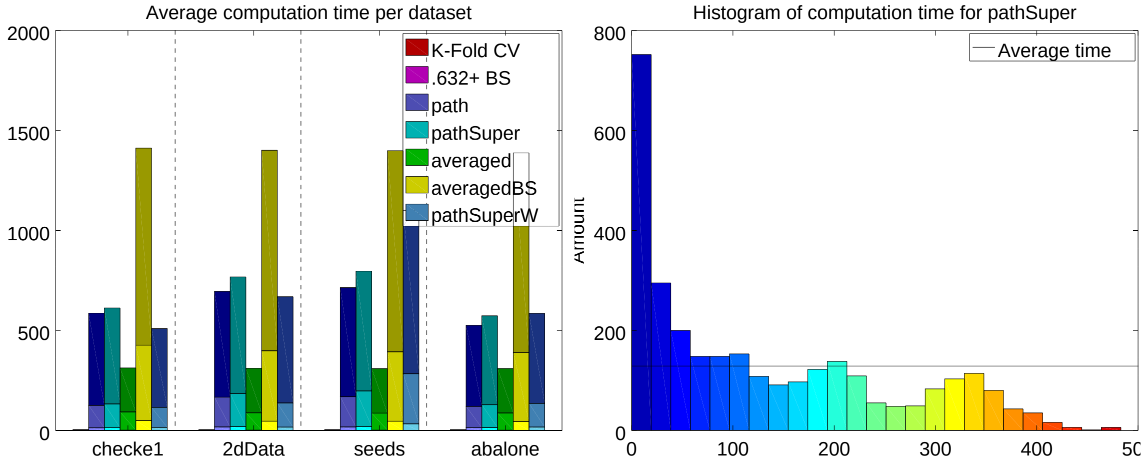

Figure 4.10: Left: Average computation times for the estimators. Right: Histogram of computation time for pathSuper

5. Conclusion and Future Work

In the scope of this work we introduced four estimators making use of both cross-validation and bootstrap techniques as well as curve fitting to estimate a classifier's performance. As part of the evaluation framework they were tested on four different datasets with three active learners and three function models and compared to both k-fold cross-validation as well as -632+ bootstrapping with regard to their error against holdout accuracy.

The results of this are somewhat mixed. On the one side, using not only the current classifier state for performance estimation appears to be a promising approach. Fitting a single learning curve as well as simulating multiple possible ways the classifier's training set could have been build both show bias and error spread similar to that of k-fold cross-validation and .632+ bootstrapping. On the other side, this comes at a hefty price; the times necessary to perform the estimation is order of magnitudes larger. The evaluation also showed that a complex function model may not be the best for fitting a learning curve, seeing as the simple linear model accomplished the lowest bias.

In the light of active learning, the path-based estimator also offers a potential advantage. In this work, it was assumed that every one of these paths to the current classifier state would be equally likely. However, using a non-random active learner like PAL, this is not true. Taking the probability of each path into account as well as using bootstrapping instead of cross-validation may lead to an improvement in classifier performance and should be looked into in further research.

A large impact also has the curve fitting itself. Using statistical weights to place higher importance on certain fitting points turned out to be a good addition, but not to the method it was expected to. Investigation into why the weights improved the path-based linear method is suggested; this in combination with a better function model could further decrease its bias.

Bibliography

[1] Airola, A., Pahikkala, T., Waegeman, W., & De Baets, B. (2011). An experimental comparison of cross-validation techniques for estimating the area under the roc curve. *Computational Statistics & Data Analysis*, *55*(4), 1828–1844. (cited on Page 11)

[2] Akaike, H. (1998). *Selected Papers of Hirotugu Akaike*, chap. Information Theory and an Extension of the Maximum Likelihood Principle, (pp. 199–213). Springer Series in Statistics. Springer New York.
URL http://link.springer.com/chapter/10.1007/978-1-4612-1694-0_15 (cited on Page 9)

[3] Archambeau, C., Valle, M., Assenza, A., & Verleysen, M. (2006). Assessment of probability density estimation methods: Parzen window and finite gaussian mixtures. In *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium*. IEEE. (cited on Page 29)

[4] Arlot, S., & Celisse, A. (2010). A survey of cross-validation procedures for model selection. *Statistics Surveys*, *4*, 40–79. (cited on Page 11)

[5] Borra, S., & Di Ciaccio, A. (2010). Measuring the prediction error. a comparison of cross-validation, bootstrap and covariance penalty methods. *Computational Statistics & Data Analysis*, *54*(12), 2976–2989.
URL http://www.sciencedirect.com/science/article/pii/S0167947310001064 (cited on Page 12)

[6] Bozdogan, H. (1987). Model selection and akaike's information criterion (aic): The general theory and its analytical extensions. *Psychometrika*, *52*(3), 345–370.
URL http://link.springer.com/article/10.1007/BF02294361 (cited on Page 9)

[7] Brumen, B., Golob, I., Jaakkola, H., Welzer, T., & Rozman, I. (2004). Early assessment of classification performance. In *ACSW Frontiers 2004, 2004 ACSW Workshops - the Australasian Information Security Workshop (AISW2004), the Australasian Workshop on Data Mining and Web Intelligence (DMWI2004), and the Australasian Workshop on Software Internationalisation (AWSI2004) . Dunedin, New Zealand, January 2004*, (pp. 91–96).
URL http://crpit.com/confpapers/CRPITV32Brumen.pdf (cited on Page 10 and 11)

[8] Chapelle, O. (2005). Active learning for parzen window classifier. In *Proceedings of the 10th International Workshop on AI and Statistics*. (cited on Page ix, 30, and 32)

[9] Charytanowicz, M., Niewczas, J., Kowalski, P. A., Kulczycki, P., Łukasik, S., & Zak, S. (2010). A complete gradient clustering algorithm for features analysis of x-ray images. *Information Technologies in Biomedicine*. (cited on Page ix, 30, and 32)

[10] Cortes, C., Jackel, L. D., Solla, S. A., Vapnik, V., & Denker, J. S. (1993). Learning curves: Asymptotic values and rate of convergence. In *Neural Information Processing Systems*.
URL http://papers.nips.cc/paper/803-learning-curves-asymptotic-values-and-rate-of-convergence.pdf (cited on Page 14)

[11] Culotta, A., & McCallum, A. (2004). Confidence estimation for information extraction. In *Proceedings of HLT-NAACL*, (pp. 109–112). (cited on Page 5)

[12] Dagan, I., & Engelson, S. P. (1995). Committee-based sampling for training probabilistic classifiers. In *Proceedings of the Twelfth International Conference on Machine Learning*, (pp. 150–157). (cited on Page 5)

[13] Dietterich, T. (1995). Overfitting and undercomputing in machine learning. *ACM Computing Surveys*, *27*(3), 326–327.
URL http://dl.acm.org/citation.cfm?id=212114 (cited on Page 8 and 21)

[14] Efron, B. (1983). Estimating the error rate of a prediction rule: Improvement on cross-validation. *Journal of the American Statistical Association*, *78*(382), 316–331.
URL http://www.cs.berkeley.edu/~jordan/sail/readings/archive/efron-improve_cv.pdf (cited on Page 12)

[15] Efron, B., & Tibshirani, R. (1997). Improvements on cross-validation: The .632+ bootstrap method. *Journal of the American Statistical Association*, *92*(438), 548–560.
URL http://www.stat.washington.edu/courses/stat527/s13/readings/EfronTibshirani_JASA_1997.pdf (cited on Page 11, 12, 13, and 22)

[16] Evans, L. P. G., Adams, N. M., & Anagnostopoulous, C. (2015). Estimating optimal active learning via model retraining improvement. *Journal of Machine Learning Research*.
URL http://arxiv.org/abs/1502.01664 (cited on Page 6)

[17] Figueroa, R. L., Zeng-Treitler, Q., Kandula, S., & Ngo, L. H. (2012). Predicting sample size required for classification performance. *BMC Medical Informatics and Decision Making*.
URL http://www.biomedcentral.com/1472-6947/12/8 (cited on Page ix, 13, 14, 15, and 25)

[18] Gupta, A. K., & Nadarajah, S. (2004). *Handbook of beta distribution and its applications*. CRC Press. (cited on Page 26)

[19] Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, 2 ed.
URL http://web.stanford.edu/~hastie/local.ftp/Springer/OLD/ESLII_print4.pdf (cited on Page 8 and 10)

[20] Joyce, J. M. (2011). *International Encyclopedia of Statistical Science*, chap. Kullback-Leibler Divergence, (pp. 720–722). Springer Berlin Heidelberg. (cited on Page 26)

[21] Kadie, C. M., & Wilkins, D. C. (1995). *Seer: Maximum Likelihood Regression for Learning-Speed Curves*. Ph.D. thesis, University of Illinois.
URL http://research.microsoft.com/en-us/um/people/carlk/papers/kadiephdfull.htm (cited on Page 14)

[22] Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence*. (cited on Page 10, 11, and 12)

[23] Kohavi, R., & Wolpert, D. H. (1996). Bias plus variance decomposition for zero-one loss functions. In *Machine Learning: Proceedings of the Thirteenth International Conference*.
URL ai.stanford.edu/~ronnyk/biasVar.pdf (cited on Page 9)

[24] Krempl, G., Kottke, D., & Lemaire, V. (2015). Optimised probabilistic active learning (opal). *Machine Learning*, *100*, 449–476. (cited on Page 30)

[25] Krempl, G., Kottke, D., & Spiliopoulou, M. (2014). Probabilistic active learning: Toward combining versatility, optimality and efficiency. In *Proceedings of the 17th International Conference on Discovery Science*. (cited on Page ix, 5, 11, 26, and 32)

[26] Krogh, A., & Vedelsby, J. (1995). Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems 7*. Massachusetts Institute of Technology.
URL http://papers.nips.cc/paper/1001-neural-network-ensembles-cross-validation-and-active-learn
pdf (cited on Page 9)

[27] Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *The Annals of Methematical Statistics*, *22*(1), 79–86.
URL http://projecteuclid.org/euclid.aoms/1177729694 (cited on Page 26)

[28] Levenberg, K. (1944). A method for the solution of certain non-linear problems. *Quarterly Journal of Applied Mathmatics*.
URL http://www.researchgate.net/publication/216212779_A_Method_for_The_Solution_of_Certain_Non-Linear_Problem_in_Least_Squares (cited on Page 20)

[29] Nash, W. J., Sellers, T. L., Cawthorn, A. J., & Ford, W. B. (1994). The population biology of abalone (haliotis species) in tasmania. i. blacklip abalone (h. rubra) from the north coast and islands of bass strait. Tech. rep., University of Calinfornia, School of Information and Computer Science. (cited on Page ix, 31, and 32)

[30] Pahikkala, T., Airola, A., Boberg, J., & Salakoski, T. (2008). Exact and efficient leave-pair-out cross validation for ranking rls. In *Proceedings of AKRR*. (cited on Page 11)

[31] Perlich, C., Provost, F., & Simonoff, J. S. (2003). Tree induction vs. logistic regression: a learning-curve analysis. *The Journal of Machine Learning Research*, *4*, 211–255.
URL http://dl.acm.org/citation.cfm?id=945375 (cited on Page 13)

[32] Rodríguez, J. D., Pérez, A., & Lozano, J. A. (2013). A general framework for the statistical analysis of the sources of variance for classification error estimators. *Pattern Recognition*, *46*(3), 855–864.
URL http://www.sciencedirect.com/science/article/pii/S0031320312003998 (cited on Page 3, 4, 7, 8, 11, and 16)

[33] Roy, N., & McCallum, A. (2001). Toward optimal active learning through monte carlo estimation of error reduction. In *Proceedings of the International Conference on Machine Learning*.
URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.137.6296 (cited on Page 6)

[34] Scheffer, T., Decomain, C., & Wrobel, S. (2001). Active hidden markov models for information extraction. In *Proceedings of the Internation Symposium on Intelligent Data Analysis*. (cited on Page 5)

[35] Schwarz, G. (1978). Estimating the dimension of a model. *The Annals of Statistics*, *6*(2), 461–464.
URL http://projecteuclid.org/euclid.aos/1176344136 (cited on Page 10)

[36] Sheather, S. T., & Jones, M. C. (1991). A reliable data-based bandwidth selection method for kernel density estimation. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, *53*(3), 683–690.
URL http://www.jstor.org/stable/2345597 (cited on Page 29)

[37] Silverman, B. W. (1986). *Density estimation for statistics and data analysis*. CRC Press. (cited on Page 29)

[38] Singh, S. (2005). Modeling performance of different classification methods: Deviation from the power law. Project Report.
URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.102.5455&rep=rep1&type=pdf (cited on Page 13, 14, and 21)

[39] van der Maaten, L. J. P. (2008). Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, (p. November). (cited on Page ix, 31, and 32)

[40] Vapnik, V. (1982). *Estimation of Dependences Based on Empirical Data*. Springer New York.
URL http://www.springer.com/us/book/9780387308654 (cited on Page 10)

[41] Weakliem, D. L. (1999). A critique of the bayesian information criterion for model selection. *Sociological Methods Research*, *27*(3), 359–397.
URL smr.sagepub.com/content/27/3/359.short (cited on Page 10)

[42] Wood, I. A., Visscher, P. M., & Mengersen, K. L. (2007). Classification based upon gene expression data: bias and precision of error rates. *Bioinformatics*, *23*(11), 1363–1370.
URL https://keppel.qimr.edu.au/contents/p/staff/CVPV127.pdf (cited on Page 13)

[43] Yelle, L. E. (1979). The learning curve: Historical review and comprehensive survey. *Decision Sciences*, *10*(2), 302–328.
URL http://onlinelibrary.wiley.com/doi/10.1111/j.1540-5915.1979.tb00026.x/abstract (cited on Page 13)

[44] Zhu, J., Wang, H., Yao, T., & Tsou, B. K. (2008). Active learning with sampling by uncertainty and density for word sense disambiguation and text classification. In *Proceedings of the 22nd International Conference on Computational Linguistics*, (pp. 1137–1144).
URL https://www.aclweb.org/anthology/C/C08/C08-1143.pdf (cited on Page 5)

[45] Zhu, X., & Goldberg, A. B. (2009). Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, *3*(1), 1–130. (cited on Page 4)

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Magdeburg, den *[. . .]*