University of Magdeburg

School of Computer Science

Bachelor Thesis

Estimating hold-out-sample Performance for Active Learning

Author:

Florian Bethe

[Month 13, 2014]

Advisors:

Prof. *[Name]*

Department of *[. . .]*

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

Abstract

[...]

Contents

List of Figures

List of Tables

List of Code Listings

1. Introduction

[...]

Goal of this Thesis

[...]

Structure of the Thesis

[...]

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 and Active Learning

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

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

or $\hat{f}(\vec{x}) = \hat{y}$ [21]. 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: $(\mathbf{X}, \mathbf{Y})$ [21]. 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 [31].

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.

Active learning is the name of a subgroup of semi-supervised methods which will be a center point of this work. To reduce the amount of labeled data needed, they choose one or more data points to be labeled, commonly with the intent to incrementally improve the built classifier's performance by adding more labeled data every iteration. Various methods to select the next data point(s) to be labeled exist; we will briefly introduce two of them: *Uncertainty sampling* and *Probabilistic Active Learning (PAL)*. The *uncertainty* of a classifier with regard to a data point describes how unsure it is about its assigned class label. This can be either seen as the distance to a *decision boundary*, that is the entity separating data points of different classes [23], or as the posterior probability estimation for the class assignment of your classifier [30]. The probabilistic approach doesn't rely on uncertainty, instead it maximizes the expected performance gain over all data point neighbourhoods [16]. A more in-depth description is given in Chapter 4.

2.2 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

$$p(X_T) = p(\vec{x}_1, y_1) \cdot ... \cdot p(\vec{x}_n, y_n) = \prod_{i=1}^{n} p(\vec{x}_i, y_i) \tag{2.2}$$

A similar formula applies for unsupervised learning. [21]

2.2.1 (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*:

$$L(f(\vec{x}), \hat{f}(\vec{x})) = \begin{cases} 0, & \text{if } f(\vec{x}) = \hat{f}(\vec{x}) \\ 1, & \text{else} \end{cases} \tag{2.3}$$

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* [12] on the training set $X_T = \{(\vec{x}_1, y_1), ..., (\vec{x}_n, y_n)\}$ is

$$Err_T = \frac{1}{N} \sum_{i=1}^{n} L(y_i, \hat{f}(\vec{x}_i)) \tag{2.4}$$

If used with the 0-1-loss function, $1 - Err$ is also known as *accuracy*: $acc = 1 - err = \frac{|CorrectClassifications|}{|TotalClassifications|}$.

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 [7].

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:

$$Err_S = E_{(\mathbf{X},\mathbf{Y})}[L(\mathbf{Y}, \hat{f}(\mathbf{X}))|X_T] \tag{2.5}$$

[21]. 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

$$Err_E = E_{(\mathbf{X},\mathbf{Y})}[L(\mathbf{Y}, \hat{f}(\mathbf{X}))] \tag{2.6}$$

Using the probability of a specific training set from Equation 2.2, 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 [12].

2.2.2 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 [15].

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:

$$\begin{aligned} Err_E &= \sum_{\vec{x}} P(\vec{x}) \left(\sigma^2(\vec{x}) + bias^2(\vec{x}) + variance(\vec{x})\right) \\ &= \frac{1}{|X|} \sum_{\vec{x}} \left(\sigma^2(\vec{x}) + bias^2(\vec{x}) + variance(\vec{x})\right) \end{aligned} \tag{2.7}$$

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 [15]. 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* [17].

2.2.3 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: $AIC = -2 \cdot log(likelihood) + 2\lambda$ [4]. 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

$$AIC = Err_T + 2\frac{\lambda}{n}\sigma^2 \tag{2.8}$$

with n as the number of training instances and σ^2 the variance of the target model [12].

A very similar estimate is the **Bayesian information criterion**(BIC). Presented by [24] 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 $log(n)$ [27]. Using the same assumptions as for AIC, it can be calculated as

$$BIC = \frac{n}{\sigma^2}[Err_T + log(n) \cdot \frac{d}{N}\sigma^2] \tag{2.9}$$

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 [12].

The third estimator for the optimism is based on the **Vapnik-Chervonenkis(VC) theory**. In their publication [26] 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:

$$sup|Err_S - Err_T| \leq 2\frac{ln(\frac{2|X_T|}{h})}{l/h} \tag{2.10}$$

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 [5].

2.2.4 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 [14].

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:

$$\widehat{Err}_{S,i} = \frac{1}{n}\sum_{j=1}^{n} L(y_{i,j}, \hat{f}(\vec{x}_{i,j})) \tag{2.11}$$

with $i = \{1, ..., k\}$ [14]. These estimations can be seen as samples of a performance distribution; examples can be uniform (simple average) or beta distribution [16]. 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 [14], while leave-pair-out seems to strike an acceptable mix of effort and accuracy [19].

An algorithm specifically designed to work with iterative labeling was introduced by [5] 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 [14]. [21] 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. [9] reinforce this argument with experiments and provide assessment of the variance as well.

2.2.5 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 [14]:

$$\widehat{Err}_i^{boot} = \frac{1}{|X_T|} \sum_{j=1}^{|X_T|} L(y_j, \hat{f}_i^b(\vec{x}_j)) \tag{2.12}$$

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)** [3]:

$$\widehat{Err}_i^{LOO} = \frac{1}{|X_T \setminus X_i^b|} \sum_{(\vec{x},y) \in X_T \setminus X_i^b} L(y, \hat{f}_i^b(\vec{x})) \tag{2.13}$$

A different bootstrap estimator was introduced by [8]. 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**:

$$\widehat{Err}^{.632} = 0.368 \cdot Err_T + 0.632 \cdot \frac{1}{b} \sum_{i=1}^{b} \widehat{Err}_i^{LOO} \tag{2.14}$$

It factors in the training error to lift up the otherwise too conservative estimate of the performance [8].

Building on that, [9] 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:

$$\hat{R} = \begin{cases} \frac{\widehat{Err}^{LOO} - Err_T}{\hat{\gamma} - Err_T}, & \text{if } \widehat{Err}^{LOO}, \hat{\gamma} > Err_T \\ 0, & \text{else} \end{cases} \tag{2.15}$$

The .632+ error rate is then defined as

$$\widehat{Err}^{.632+} = \widehat{Err}^{.632} + \left(min(\widehat{Err}, \hat{\gamma}) - Err_T\right) \cdot \frac{0.368 \cdot 0.632 \cdot \hat{R}}{1 - 0.368 \cdot \hat{R}} \tag{2.16}$$

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 $\mathbf{X}$ and $\mathbf{Y}$ and a class probability of 0.5 in a 2-class problem, whereas the .632 does not [9]. However, [28] 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.

2.3 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 [29]. 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 [20].

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.

2.3.1 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 [11]. An example of such a curve can be seen in figure Figure 2.1. Functions families fitting this description are **power** [11, 25], **logarithmic**

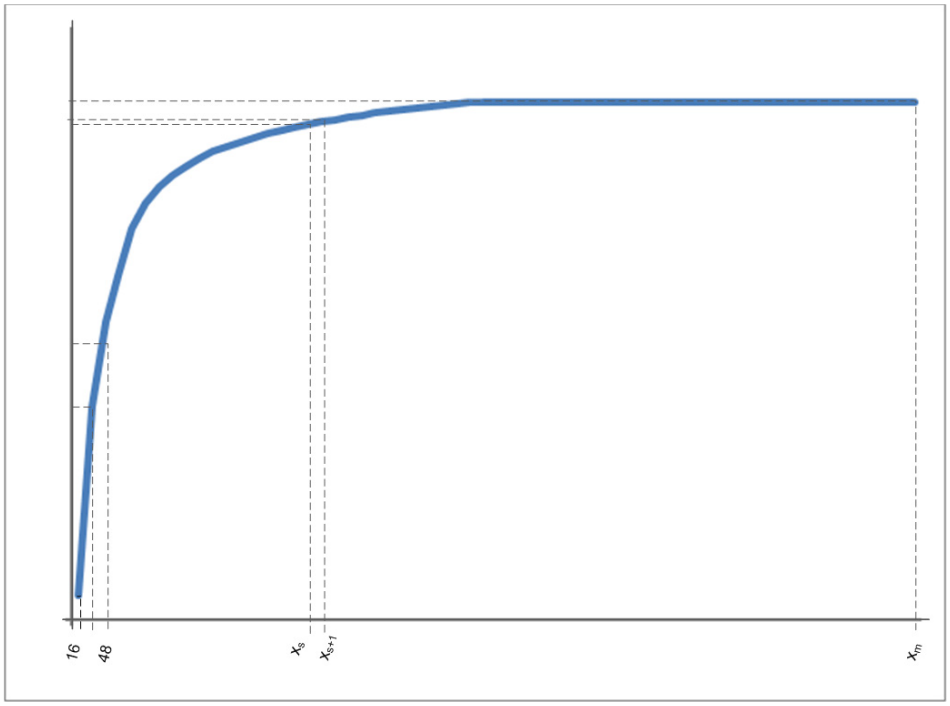

Figure 2.1: General appearance of a learning curve. Training set size as x-axis, accuracy as y-axis [11]

and **exponential** [25]. These papers provide an overview for the fitting of the function types. [25] 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 [11]. 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 [6]. 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 Miscellaneous

Introduced in [10], **Model Retraining Improvement** is originally intended as a statistically optimal criteria for instance selection in active learning. However, it does so by performing an unbiased estimate of the loss improvement a new labeled instance would yield. While the algorithm itself does not provide a way to estimate the future loss (or current, either), the paper provides a theoretical background for potential future estimators.

Similar to the extrapolation of fitted learning curves is the *maximum-likelihood-regression* presented in [13]. While the former has a fixed model to regress on, their method finds the most probable model for the given data.

[22] also presents a method to estimate current and future loss of a classifier in an active learning setting, similarly to model retraining improvement. They, however, use *Monte-Carlo-Sampling*, resulting in a process comparable to the techniques already discussed in this chapter.

3. Proposed Methods

As described in the previous section, current performance estimation methods can be roughly categorized into two sections. One group uses information present for the current state of a classifier, e.g. the cross-validation methods. The second, much smaller group takes a look at the performance development over time, up to the current iteration. This encompasses the fitting of function models expected to be of similar shape to the learning curve using already present performance estimates to extrapolate them to future iterations. Typical methods for the estimation of the already witnessed iterations include holdout testing and k-fold cross-validation [11].

The methods proposed in this work focus on combining both groups in an attempt to use as much information as possible to increase the prediction quality; .632 bootstrapping and the likes focus on the set of available instances, ignoring the process that led to this state. In turn, curve fitting currently makes use of either subpar techniques to obtain the accuracies for each iteration or uses holdout samples which can be costly or not available at all.

To serve as an illustration of the thought process leading to the following techniques, we assume, without loss of generality, a dataset $D = (\vec{x_i}, y_i)$ with two class labels $y_i \in 0, 1$ and $|D| = n$. From this set an active learner selects $k \leq n$ instances which serve as the training set X_T of a classifier $c : \vec{x} \mapsto y$. We would like to know the prediction accuracy of c for the set D.

3.1 Performance estimation on training sub-sets

The simplest way to obtain performances estimates for the process of instance selection would be to use *leave-x-out(LXO) cross-validation*, with $x \in \{1, ..., k-1\}$. This way, we obtain $\binom{k}{x}$ subsets $S_{T,i}$ of size x from the original training set as well as corresponding test sets of size $k-x$, which enable the performance estimation. Both low computational cost for the individual estimates as well as unbiasedness [21] make it a reasonable

choice. While the computational effort for each subset is low, the total amount of possible subsets is $2^k - 2$, only leaving out the edge cases with size 0 and k, resulting in exponential complexity if used natively. Thus, some kind of sampling to reduce the complexity seems desirable.

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. This comes at a price, however; additional computational effort for the creation and testing of the bootstrap samples is necessary. Also, it isn't trivial how to create and handle the subsets. One could proceed similar to its cross-validation counterpart, sampling from the training set without replacement and using that subset as base for the bootstrap samples. This has two obvious downsides: for one, we are unable to obtain performance estimates for set sizes of one (except for naïve bootstrap, which doesn't require that test instances are not within the bootstrap sample). To solve this, the instances not selected for the subset could be used as (additional) test sets, as they are when using cross-validation. However, the influence this would have on the .632 family is unclear; although theoretically additional test instances should not revoke the name-giving necessity of weighting between training and bootstrap performance, the weights themselves may be incorrect, as the bootstrap would be tested on data not available to the training performance estimation. Of course, a possibility could be to also test the training performance on those additional instances, but then it wouldn't be the training error anymore, making it even more unclear.

3.1.1 Sub-sampling of fitting points

Regardless of which estimation technique is used, the complexity still scales exponentially with the set size k. To reduce the amount of estimates for the fitting process, some sort of selection has to occur. In this work, three sub-sampling strategies were explored. Reducing the information available naturally has some drawbacks, including an expected higher variance and, if done improper, an added bias. Also, the sampling may influence the fitting itself, potentially inflicting additional penalties to the robustness.

A simple, yet effective method is to cap the number of estimates. Possible options are to either impose a fixed, hard cap for all training set sizes $|X_T|$, or to use a polynomial dependent on k, e.g. k^2. A potential pitfall is the selection of the subsets $S_{T,i} \subset X_T$ to be evaluated. Selecting either randomly over all possible subsets or from pools for each subset size $|S_{T,i}| \in \{1, ..., k-1\}$ with sizes proportional to $\binom{|S_{T,i}|}{k}$ prevents unintentional importance assignment to subset sizes.

A related approach exports the computational cost to the fitting process. Instead of selecting multiple subsets per size once and using them for fitting, we select only one subset per size multiple times and fit on them separately. Formally, we have a tuple $\tilde{S}_j = (S_1, ..., S_{k-1})$ with $S_i \subset X_T$, $|S_i| = i$ and $j = \{1, ..., r\}$. 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 without replacement. 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 grow in size of the training set in turn causes a reduction in size of the test set, the amount of potential outcomes shrinks from k to 2 for $|S_T| = \{1, ..., k-1\}$. Thus, the number of unique combinations would shrink to $k!$.

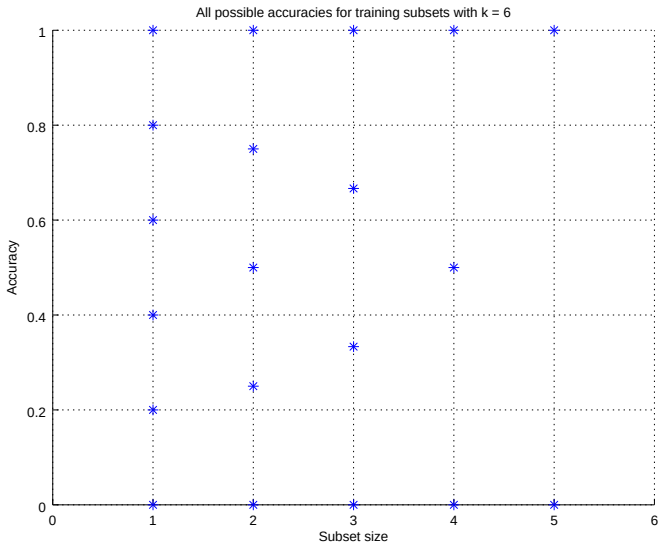

Figure 3.1: All possible accuracies for training subsets with $k = 6$

Unfortunately, in order to use the estimations as fitting data, they would have to be computed first; we need to know the distribution of accuracies for each subset size to weight them properly. Otherwise, a uniform weighting (or randomly picking one) would simply lead to a prediction of 0.5 and we would gain nothing. Thus, using the estimates themselves is not feasible as a reduction of estimation samples, but still limits the number of curves that have to be fit, as duplicates do not have to be fit twice; counting it twice for the final estimation suffices.

Taking a second look at the aforementioned method, something seems to be odd. As it is, randomly taking subsets of each size allows for something counterintuitive to happen: instances present in subsets of smaller size do not necessarily have to be selected for their larger brethren. 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. This again reduces the possible combinations

to $k!$, as the sampling is now done without replacement. This time, however, we do not rely on the final accuracy distribution, as each path is equally likely, meaning that the 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.

3.1.2 Grouping of performance estimates

Now that the subsets have been selected, the individual performances have to be estimated. For the cross-validation approach, this requires a classifier being trained on each of them and testing it on the remaining instances $X_T \setminus S_i$. Next in line is the decision which estimates to use for the fitting, and how. Depending on the sub-sampling selected, different options are available:

- **All data**: The performance estimates are combined with their respective subset size and then used to fit a single curve.
- **Averaged data**: As originally intended for leave-x-out, the estimates are first arithmetically averaged over their respective subset size, then used to fit a single curve. This has the side effect that it skews the "importance" of the estimates, as more of them are available for middle-of-the-pack subset sizes. By not using all of them, the inherent weighting is removed.
- **One path at a time**: For the path-based sub-sampling, we have only one estimate per subset size anyway, but multiple times. Consequently, we fit one curve for each path. This results in multiple curves and thus multiple final performance estimates.

3.2 Combining sub-estimates with curve fitting

3.2.1 Function models and fitting algorithms

As a big part of the fitting process, some thought has to go into the selection of an appropriate function model. Not only does it have to be capable of modeling the learning process, it 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 [18]. 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 minima 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 [25]. A good candidate for least squares fitting seems to be the 3-parameter exponential law

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

Unfortunately, it falls in the category "non-linearizable" and requires iterative fitting. A different non-linearizable function class are sigmoids. For the evaluation, we also use a sigmoid of the form

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

While they weren't tested 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.

3.2.2 Fitting improvements

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 a study questioning people about the value of the elementary electric charge. Averaging the individual statements may, with a sufficiently large sample size, give an okay approximation to the real value. However, if the answers of people related to physics (students, engineers etc.) are counted twice, the approximation is likely to be better, as these people are more likely to know the correct answer.

To transfer this to our problem, we take a second look at the averaging grouping method. Here, we average the estimates and fit on them instead. This bears the problem of removing the information that for some fitting points more estimates were present.

Using statistical weights in the fitting process proportional to the original amount of estimations brings this back into the fitting process, leading to correct importance of the data points.

Another potential improvement for the fitting process is the addition of data. Of course, while using the subset method, we cannot simply get more data without purchasing more labels. But we can get information for something that isn't covered by that method. Showcased in [9] .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 2.15). And while we do have performance estimates for each subset size $|S_T| \in \{1, ..., k-1\}$, we are missing an estimate for $|S_T| = 0$. The no-information rate may be used as the 0th fitting point, filling that gap.

4. Evaluation

[...]

5. Conclusion and Future Work

[...]

A. Appendix

[...]

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 9)

[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 7)

[3] 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 9)

[4] 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 7)

[5] 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 8 and 9)

[6] 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 12)

[7] 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 6)

[8] 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 9 and 10)

[9] 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 9, 10, and 18)

[10] 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 12)

[11] 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 vii, 11, 12, and 13)

[12] 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 5, 6, 7, and 8)

[13] 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 12)

[14] 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 8 and 9)

[15] 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 6 and 7)

[16] 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*.
URL https://kmd.cs.ovgu.de/teaching/hli/KremplKottkeSpiliopoulou2014DS_FULL.pdf (cited on Page 4 and 8)

[17] 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 7)

[18] 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 16)

[19] 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 9)

[20] 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 11)

[21] 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, 5, 6, 9, and 13)

[22] 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 12)

[23] 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 4)

[24] 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 7)

[25] 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 11, 12, and 17)

[26] Vapnik, V. (1982). *Estimation of Dependences Based on Empirical Data*. Springer New York.
URL http://www.springer.com/us/book/9780387308654 (cited on Page 8)

[27] 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 7)

[28] 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 10)

[29] 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 11)

[30] 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 4)

[31] 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 *[. . .]*