University of Magdeburg

School of Computer Science

Bachelor Thesis

[The Title of the Thesis]

Author:

[Forename] [Surname]

[Month 13, 2014]

Advisors:

Prof. *[Name]*

Department of *[...]*

***[Surname]*, *[Forename]*:**
[The Title of the Thesis]
Bachelor Thesis, University of Magdeburg, *[2014]*.

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 our 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 low" 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*: $\vec{x} = (f_{a_1,1}, ..., f_{a_n,n})$ with $f_{i,j} \in F_j \; \forall i = \{1, ..., |F_j|\}, j = \{1, ..., n\}$ and F_j 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 $\hat{f}(\vec{x}) = \hat{y}$[RPL13]. 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 all expected data and should be sampled 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: $(\mathbf{X}, \mathbf{Y})$ [RPL13]. 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). It is similar to the process of density estimation from statistics.

- **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 [ZG09].

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 [SDW01], or as the posterior probability estimation for the class assignment of your classifier [ZWYT08].

The probabilistic approach doesn't rely on uncertainty, instead it maximizes the expected performance gain in all data point neighbourhoods [KKS14]. 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 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

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

[RPL13]. A similar formula applies for unsupervised learning.

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

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

[HTF09]. 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 [Die95].

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:

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

[RPL13]. 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

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

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 specific classifier but can guide the selection of a preferable algorithm. Note that we still require knowledge of the underlying distributions for a direct evaluation, realistically making an estimator necessary as well [HTF09].

2.2.2 Bias and variance

In the previous sections we portrayed classifier as a simple mapping $\hat{f}$ of input values $\vec{x}$ to class labels y. While this is true for many of them like decision trees, a more general point of view is to see a classifier as an assignment of probabilities for the 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. For classifiers of the previously described type, only one class label has a non-zero probability for 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 anyway [KW96].

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 [KW96]. 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*.

2.2.3 Classificator-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 two methods to estimate the optimism: *Akaike information criterion* and *Bayes information criterion*.

The **Akaike information criterion**(AIC) in its original form is defined as a function of a model's likelihood and complexity λ: $AIC = -2 \cdot log(likelihood) + 2\lambda$ [Boz87]. 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 [HTF09].

A very similar estimate is the **Bayesian information criterion**(BIC). Defined as $BIC = -2 \cdot log(likelihood) + \lambda log(n)$, it is asymptotically equal to AIC, but with a steeper penalty for complex models due to the exchange of the factor 2 with $log(n)$ [Wea99]. Using the same assumptions as for AIC, we get the formula

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

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 [HTF09].

2.2.4 Cross-Validation

2.2.5 Bootstrapping

2.3 Learning Curves and Regression Models

2.3.1 Function families

2.3.2 Algorithms

3. Example Chapter

This chapter gives you some examples how to include graphics, create tables, or include code listings. But first, we start with a short description how you can efficiently cite in LaTeX. The following footnote shows you how to reference URLs and where this document is available online.[1]

3.1 Citation

There are several types of literature. The most citations are workshop and conference papers. Please use the inproceedings-tag for those citations (e.g., [**?**]). You should have short-hands for workshop and conference names to be sure the naming is consistent and uniform (see our BibTeX files how to do that).

Slightly different are articles published in journals (e.g., [**?**]). Make sure you that the volume and number-tags are present and that no inproceeding is tagged as article or vice versa.

You might want to take a look at the example BibTeX file to find out how to cite books [**?**], technical reports [**?**], websites [**?**], PhD theses, or master theses [**?** **?**].

3.2 Formulas

There are different types of mathematical environments to set formulas. The equation $E = m \cdot c^2$ is an inline formula. But you can also have formulas at a separate line (see Equation 3.1).

$$P = \big(\mathcal{A} \Rightarrow (\mathcal{B} \Leftrightarrow \mathcal{C}) \wedge (\mathcal{B} \Leftrightarrow \mathcal{D})\big) \wedge (\mathcal{B} \Rightarrow \mathcal{A}) \wedge (\mathcal{C} \Rightarrow \mathcal{A}) \wedge (\mathcal{D} \Rightarrow \mathcal{A}) \tag{3.1}$$

[1] http://www.ovgu.de/tthuem

If you need multiple lines that are aligned to each other, you might want to use the following code.

$$
\begin{aligned}
& \text{GraphLibrary} \\
\wedge\ & (\text{GraphLibrary} \Rightarrow \text{Edges}) \wedge (\text{Edges} \vee \text{Algorithms} \Rightarrow \text{GraphLibrary}) \\
\wedge\ & (\text{Edges} \Leftrightarrow \text{Directed} \vee \text{Undirected}) \wedge (\neg\text{Directed} \vee \neg\text{Undirected}) \\
\wedge\ & (\text{Algorithms} \Leftrightarrow \text{Number} \vee \text{Cycle}) \\
\wedge\ & (\text{Cycle} \Rightarrow \text{Directed}).
\end{aligned}
$$

3.3 Graphics

In Figure 3.1, we give a small example how to insert and reference a figure.

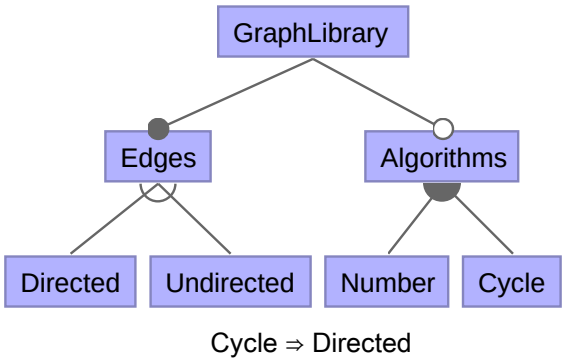

Figure 3.1: A feature model representing a graph product line

3.4 Tables

Table 3.1 shows the result of a simple tabular environment.

Group Type	Propositional Formula
And	$(P \Rightarrow C_{k_1} \wedge \ldots \wedge C_{k_m}) \wedge (C_1 \vee \ldots \vee C_n \Rightarrow P)$
Or	$P \Leftrightarrow C_1 \vee \ldots \vee C_n$
Alternative	$(P \Leftrightarrow C_1 \vee \ldots \vee C_n) \wedge \text{atmost1}(C_1, \ldots, C_n)$

Table 3.1: Mapping a feature model to a propositional formula

3.5 Code Listings

In Listing 3.1 on the next page, we give an example of a source code listing.

```
1  class A extends Object {
2      A() { super(); }
3  }
4  class B extends Object {
5      B() { super(); }
6  }
7  class Pair extends Object {
8      Object fst;
9      Object snd;
10     Pair(Object fst, Object snd) {
11         super(); this.fst=fst; this.snd=snd;
12     }
13     Pair setfst(Object newfst) {
14         return new Pair(newfst, this.snd);
15     }
16 }
```

Listing 3.1: Java source code

4. Evaluation

[...]

5. Related Work

Stuff

6. Conclusion

[...]

7. Future Work

[...]

A. Appendix

[...]

Bibliography

[Boz87] Hamparsum Bozdogan. Model selection and akaike's information criterion (aic): The general theory and its analytical extensions. *Psychometrika*, 52(3):345–370, September 1987. (cited on Page 7)

[Die95] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM Computing Surveys*, 27(3):326–327, September 1995. (cited on Page 6)

[HTF09] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer, 2 edition, 2009. (cited on Page 5, 6, and 7)

[KKS14] Georg Krempl, Daniel Kottke, and Myra Spiliopoulou. Probabilistic active learning: Toward combining versatility, optimality and efficiency. In *Proceedings of the 17th International Conference on Discovery Science*, October 2014. (cited on Page 5)

[KW96] Ron Kohavi and David H. Wolpert. Bias plus variance decomposition for zero-one loss functions. In *Machine Learning: Proceedings of the Thirteenth International Conference*, 1996. (cited on Page 6 and 7)

[RPL13] Juan D. Rodríguez, Aritz Pérez, and Jose A. Lozano. A general framework for the statistical analysis of the sources of variance for classification error estimators. *Pattern Recognition*, 46(3):855–864, March 2013. (cited on Page 3, 4, 5, and 6)

[SDW01] Tobias Scheffer, Christian Decomain, and Stefan Wrobel. Active hidden markov models for information extraction. In *Proceedings of the Internation Symposium on Intelligent Data Analysis*, 2001. (cited on Page 4)

[Wea99] David L. Weakliem. A critique of the bayesian information criterion for model selection. *Sociological Methods Research*, 27(3):359–397, February 1999. (cited on Page 7)

[ZG09] Xiaojin Zhu and Andrew B Goldberg. Introduction to semi-supervised learning. *Synthesis lectures on artificial intelligence and machine learning*, 3(1):1–130, 2009. (cited on Page 4)

[ZWYT08] Jingbo Zhu, Huizhen Wang, Tianshun Yao, and Benjamin K. Tsou. 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*, pages 1137–1144, August 2008. (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 *[. . .]*