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 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 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: $(\mathbf{X}, \mathbf{Y})$ [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]. 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 [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* [KV95].

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 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 [Aka98] in 1973. It 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). Presented by [Sch78] 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 $log(n)$ [Wea99]. 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 [HTF09].

The third estimator for the optimism is based on the **Vapnik-Chervonenkis(VC) theory**. In their publication [Vap82] 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 [BGJ+04].

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 [Koh95].

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 $i = \{1, ..., k\}$ [Koh95]. These estimations can be seen as samples of a performance distribution; examples can be uniform (simple average) or beta distribution [KKS14]. 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 [Koh95], while leave-pair-out seems to strike an acceptable mix of effort and accuracy [PABS08].

An algorithm specifically designed to work with iterative labeling was introduced by [BGJ+04] 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. [APWDB11] 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 [Koh95]. [RPL13] 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 bias, overestimating the error rate. [ET97] reinforce this argument with experiments and provide assessment for 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:

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

[Koh95].

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 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:

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

, a technique known as **leave-one-out bootstrapping** or **LOO** [BDC10].

A different bootstrap estimator was introduced by [Efr83]. 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 [Efr83].

Building on that, [ET97] developed a modified .632 bootstrap under the namse 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 [ET97]. However, [WVM07] show in their paper about different error rate estimators that leave-one-out slightly underestimates the error. Unfortunately, no experiments with .632+ were performed to evaluate the influence of this.

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 [Yel79]. 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 [PPS03].

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 [FZTKN12]. An example of such a curve can be seen in figure Figure 2.1. Functions families fitting this description are **power** [FZTKN12,

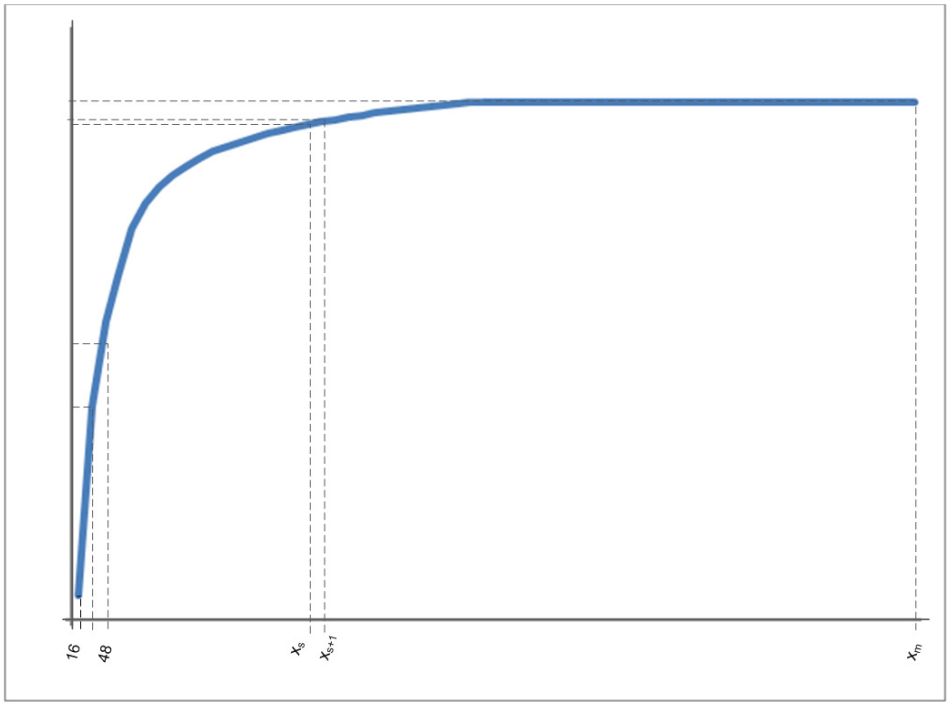

Figure 2.1: General appearance of a learning curve. Training set size as x-axis, accuracy as y-axis [FZTKN12]

Sin05], **logarithmic** and **exponential** [Sin05]. These papers provide an overview for the fitting of the function types. [Sin05] 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 [FZTKN12]. 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 [CJS+93]. 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 [EAA15], **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 [KW95]. While the former has a fixed model to regress on, their method finds the most probable model for the given data.

[RM01] 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. 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

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).

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

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

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

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

3.3 Graphics

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

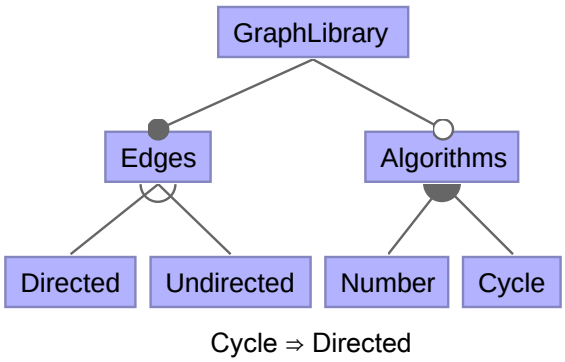

Figure 3.1: A feature model representing a graph product line

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

[Aka98] Hirotugu Akaike. *Selected Papers of Hirotugu Akaike*, chapter Information Theory and an Extension of the Maximum Likelihood Principle, pages 199–213. Springer Series in Statistics. Springer New York, 1998. (cited on Page 7)

[APWDB11] Antti Airola, Tapio Pahikkala, Willem Waegeman, and Bernard De Baets. An experimental comparison of cross-validation techniques for estimating the area under the roc curve. *Computational Statistics & Data Analysis*, 55(4):1828–1844, April 2011. (cited on Page 9)

[BDC10] Simone Borra and Agostino Di Ciaccio. Measuring the prediction error. a comparison of cross-validation, bootstrap and covariance penalty methods. *Computational Statistics & Data Analysis*, 54(12):2976–2989, December 2010. (cited on Page 9)

[BGJ+04] Bostjan Brumen, Izidor Golob, Hannu Jaakkola, Tatjana Welzer, and Ivan Rozman. 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*, pages 91–96, 2004. (cited on Page 8 and 9)

[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)

[CJS+93] Corinna Cortes, L. D. Jackel, Sara A. Solla, Vladimir Vapnik, and John S. Denker. Learning curves: Asymptotic values and rate of convergence. In *Neural Information Processing Systems*, 1993. (cited on Page 12)

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

[EAA15] Lewis P. G. Evans, Niall M. Adams, and Christoforos Anagnostopoulous. Estimating optimal active learning via model retraining improvement. *Journal of Machine Learning Research*, 2015. (cited on Page 12)

[Efr83] Bradley Efron. Estimating the error rate of a prediction rule: Improvement on cross-validation. *Journal of the American Statistical Association*, 78(382):316–331, 1983. (cited on Page 9 and 10)

[ET97] Bradley Efron and Robert Tibshirani. Improvements on cross-validation: The .632+ bootstrap method. *Journal of the American Statistical Association*, 92(438):548–560, June 1997. (cited on Page 9 and 10)

[FZTKN12] Rosa L. Figueroa, Qing Zeng-Treitler, Sasikiran Kandula, and Long H. Ngo. Predicting sample size required for classification performance. *BMC Medical Informatics and Decision Making*, 2012. (cited on Page vii and 11)

[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, 7, and 8)

[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 and 8)

[Koh95] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *International Joint Conference on Artificial Intelligence*, 1995. (cited on Page 8 and 9)

[KV95] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in Neural Information Processing Systems 7*. Massachusetts Institute of Technology, 1995. (cited on Page 7)

[KW95] Carl Myers Kadie and David C. Wilkins. *Seer: Maximum Likelihood Regression for Learning-Speed Curves*. PhD thesis, University of Illinois, 1995. (cited on Page 12)

[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)

[PABS08] Tapio Pahikkala, Antti Airola, Jorma Boberg, and Taipo Salakoski. Exact and efficient leave-pair-out cross validation for ranking rls. In *Proceedings of AKRR*, 2008. (cited on Page 9)

[PPS03] Claudia Perlich, Foster Provost, and Jeffrey S. Simonoff. Tree induction vs. logistic regression: a learning-curve analysis. *The Journal of Machine Learning Research*, 4:211–255, December 2003. (cited on Page 11)

[RM01] Nicholas Roy and Andrew McCallum. Toward optimal active learning through monte carlo estimation of error reduction. In *Proceedings of the International Conference on Machine Learning*, 2001. (cited on Page 12)

[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, 6, and 9)

[Sch78] Gideon Schwarz. Estimating the dimension of a model. *The Annals of Statistics*, 6(2):461–464, 1978. (cited on Page 7)

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

[Sin05] Sameer Singh. Modeling performance of different classification methods: Deviation from the power law. Project Report, 2005. (cited on Page 11)

[Vap82] Vladimir Vapnik. *Estimation of Dependences Based on Empirical Data*. Springer New York, 1982. (cited on Page 8)

[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)

[WVM07] Ian A. Wood, Peter M. Visscher, and Kerrie L. Mengersen. Classification based upon gene expression data: bias and precision of error rates. *Bioinformatics*, 23(11):1363–1370, June 2007. (cited on Page 10)

[Yel79] Louis E. Yelle. The learning curve: Historical review and comprehensive survey. *Decision Sciences*, 10(2):302–328, April 1979. (cited on Page 10)

[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 *[. . .]*