Novel Learning Tasks From Practical Applications

Ralf Klinkenberg*, **Oliver Ritthoff***, **and Katharina Morik***

* University of Dortmund, Department of Computer Science, Chair of Artificial Intelligence, 44221 Dortmund, Germany, E-Mail: {klinkenberg,ritthoff,morik}@ls8.cs.uni-dortmund.de

Abstract. Some classes of learning problems have been well-posed and investigated, especially the ones of *classification* and *regression*. However, in practice we are often confronted with modified learning tasks that deviate from these standard scenarios. In other words, given an application problem, the assumptions made when treating it as a standard learning task are often not appropriate. The consequence of an inadequate problem representation is, that either the learning performance in terms of accuracy decreases, or some of the target hypotheses simply cannot be learned. In this paper we present several nonstandard learning tasks, comprising *learning drifting concepts*, *transductive learning*, *learning with prior knowledge*, and *feature generation and selection*, handling different kinds of representation inadequacies. The corresponding studies have been conducted in connection with the collaborative research center on computational intelligence (SFB 531).

Keywords. learning task, concept drift, transduction, learning with prior knowledge, feature selection and generation, machine learning environment

1 Introduction

Some classes of learning problems have been well-posed and investigated. The task that has been understood best is the one of concept learning (*classification*): Given training examples X in a representation language L_E, which are classified according to an unknown Boolean function $c(x)$ and a space of hypotheses L_H, learn a Boolean-valued function $h \in L_H$, such that $h(x) = c(x)$ for all x. The quality of the learning result is evaluated with respect to a test set of examples and a quality measure, most often accuracy ($h(x) = c(x)$) and coverage (all x are handled by $h(x)$). If the function to be learned is real-valued, the learning task is called *regression*.

Deviations from the standard learning tasks in learnability research are driven by the aim of determining the exact border between polynomially learnable and not learnable concepts. In this paper we outline some results achieved in our project "Identification and Formalization of Non-standard Learning Tasks From Practical Applications" (B5) within the collaborative research center "Design and Management of Complex Technical Processes and Systems by Means of Computational Intelligence Methods" (SFB 531) studying non-standard learning tasks that are *driven by real-world problems* [4].

In practice we often meet modified learning tasks that deviate from the standard scenario of classification and regression. In other words, given an application problem, the assumptions made when treating it as a standard learning task are often not appropriate. As a consequence of this inappropriateness, either the learning performance in terms of the predictive error (accuracy) decreases, or some hypotheses simply cannot be learned.

In many real-world applications, the concept to be learned changes over time. In reality we cannot always assume a concept to be given once and for all. The distribution of examples regarding the labels is not always constant, but often changes over time. The corresponding non-standard learning task is called *learning drifting concepts*.

It is not necessary for real-world problems to be more difficult than the standard problems. Sometimes, we are given more knowledge than the standard learning tasks assume. Vapnik has raised the question, why we should induce a hypothesis on the basis of examples first and then apply our learning result to a test set, if the given test set is all of our problem. This means that sometimes we know in advance the position or attributes of the test examples that we have to classify. If this is so, why not ease the learning task by directly approaching the test set? The non-standard learning task of learning from labeled as well as unlabeled examples is called *transduction*.

Another piece of information a user of machine learning methods may have is the importance or misclassification cost of particular examples. While the standard tasks consider all examples of equal worth and achievability, in medicine, for instance, some critical observations can be made only rarely, but are considered very important. If a patient's state can be measured by a particular catheter and this catheter is only seldomly applied, we want to strengthen the influence of its measurements on the learning result. As opposed to background knowledge (i.e., a theory), *prior knowledge* about examples can ease learning.

In cases where the example language L_E is not expressive enough, the correct hypothesis cannot be found by the learner. If, for instance, a learning algorithm can only separate data linearly, but the positive examples are separated by a curve (e.g., a polynomial) from the negative examples, the target concept cannot be determined. A mapping from the original feature space to another one may shape the examples, such that the learner can recognize the target concept. The transformation of L_E in order to ease learning is called *feature generation*. A contrary situation occurs, if the example language contains too many irrelevant or redundant features. This often degrades the learning performance. Intuitively, we can imagine that the true concept is hidden in the noise of irrelevant descriptions. A solution to this problem is *feature selection*, i.e., the process of removing redundant or irrelevant features from the original feature set.

1.1 Overview

In the following section we give a short introduction into the principles of support vector machines (SVM) (Section 2), since, due to its positive theoretical and practical properties, this learning method was intensively used in our experiments. Subsequently, we briefly present results achieved by studying different non-standard learning tasks: learning drifting concepts (Section 3), transduction (Section 4), learning with prior knowledge (Section 5), and feature generation and selection (Section 6). To study these non-standard learning tasks a flexible learning environment was needed. A short description of an environment that fulfills the necessary requirements is given in Section 7. Finally, Section 8 summarizes the most important results of our work and gives a short outlook to open research topics in the field of non-standard learning tasks.

2 Support Vector Machines

Support vector machines are based on the principle of structural risk minimization (SRM) [32] from statistical learning theory. In their basic form, SVM learn linear decision rules,

$$h(\vec{x}) = sign\{\vec{w} \cdot \vec{x} + b\} = \begin{cases} +1, & \text{if } \vec{w} \cdot \vec{x} + b > 0 \\ -1, & \text{else} \end{cases}$$

described by a weight vector $\vec{w}$ and a threshold b. The idea of structural risk minimization is to find a hypothesis h for which one can guarantee the lowest probability of error. For SVM, [32] shows that this goal can be translated into finding the hyperplane with maximum soft-margin. Computing this hyperplane is equivalent to solving the following optimization problem:

Optimization Problem 1 (SVM (primal))

$$\textit{Minimize:} \quad V(\vec{w}, b, \vec{\xi}) = \frac{1}{2}\vec{w} \cdot \vec{w} + C \sum_{i=1}^{n} \xi_i \quad (1)$$

$$\textit{subject to:} \quad \forall_{i=1}^{n} : y_i[\vec{w} \cdot \vec{x}_i + b] \geq 1 - \xi_i \quad (2)$$

$$\forall_{i=1}^{n} : \xi_i \geq 0 \quad (3)$$

In this optimization problem, the Euclidean length $||\vec{w}||$ of the weight vector is inversely proportional to the soft-margin of the decision rule. The constraints (2) require that all training examples are classified correctly up to some slack ξ_i. If a training example lies on the "wrong" side of the hyperplane, the corresponding ξ_i is greater or equal to 1. Therefore $\sum_{i=1}^{n} \xi_i$ is an upper bound on the number of training errors. The factor C in (1) is a parameter that allows trading-off training error vs. model complexity.

For computational reasons, it is useful to solve the Wolfe dual of optimization problem 1 instead of solving optimization problem 1 directly [32].

Optimization Problem 2 (SVM (dual))

$$\textit{Minimize:} \; W(\vec{\alpha}) = -\sum_{i=1}^{n}\alpha_i + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j (\vec{x}_i \vec{x}_j)$$

$$\textit{subject to:} \; \sum_{i=1}^{n} y_i \alpha_i = 0$$

$$\forall_{i=1}^{n} : 0 \leq \alpha_i \leq C$$

Joachims has developed a fast algorithm for computing the solution to this optimization problem [9, 12]. We use this algorithm implemented in SVM^{light} for the classification and transduction[1] experiments described in this paper.[2]

[1] The learning task of transduction and the corresponding optimization problems are described in Section 4.

[2] SVM^{light}: available at *http://svmlight.joachims.org/*

Support vectors are those training examples $\vec{x}_i$ with $\alpha_i > 0$ at the solution. From the solution of optimization problem 2, the decision rule can be computed as

$$\vec{w} \cdot \vec{x} = \sum_{i=1}^{n} \alpha_i y_i (\vec{x}_i \cdot \vec{x}) \quad \text{and} \quad b = y_{usv} - \vec{w} \cdot \vec{x}_{usv} \ .$$

The training example $(\vec{x}_{usv}, y_{usv})$ for calculating b must be a support vector with $\alpha_{usv} < C$. Finally, the training losses ξ_i can be computed as

$$\xi_i = max(1 - y_i \left[\vec{w} \cdot \vec{x}_i + b\right], 0).$$

For solving optimization problem 2 as well as applying the learned decision rule, it is sufficient to be able to calculate inner products between feature vectors. Exploiting this property, kernels $\mathcal{K}(\vec{x}_1, \vec{x}_2)$ for learning nonlinear decision rules can be introduced. Depending on the type of kernel function, SVM learn polynomial classifiers, radial basis function (RBF) classifiers, or two layer sigmoid neural nets. Such kernels calculate an inner product in some feature space and replace the inner product in the formulas above.

3 Learning Drifting Concepts

Most machine learning approaches assume that the distribution underlying the training examples and new unseen test examples is static, and that a model, once learned on the training data, can be applied to any new test example and does not need any adjustments later on. For many real-world machine learning tasks, however, where data is collected over an extended period of time, its underlying distribution is likely to change over time. A typical example is information filtering, i.e., the adaptive classification of text documents with respect to a particular user interest. Information filtering techniques are used, for example, to build personalized news filters, which learn about the news-reading preferences of a user or to filter e-mail. Both the interest of the user and the document content change over time. A filtering system should be able to adapt to such concept changes.

This section describes a method first proposed in [17] to recognize and handle concept changes with support vector machines. The method maintains a window on the training data. The key idea is to automatically adjust the window size so that the estimated generalization error is minimized. The approach is both theoretically well founded as well as effective and efficient in practice. Since it does not require complicated parameterization, it is simpler to use and is more robust than comparable heuristics. Experiments with simulated concept drift scenarios based on real-world text data compare the method with other window management approaches. We show that it can effectively select an appropriate window size in a robust way.

The problem of concept drift for the pattern recognition problem can be formalized in the following framework. Each example $\vec{z} = (\vec{x}, y)$ consists of a feature vector $\vec{x} \in \mathbb{R}^{\mathbb{N}}$ and a label $y \in \{-1, +1\}$ indicating its classification. Data arrives over time in batches. Without loss of generality these batches are assumed to be of equal size, each containing m examples:

$$\vec{z}_{(1,1)}, \ldots, \vec{z}_{(1,m)}, \vec{z}_{(2,1)}, \ldots, \vec{z}_{(2,m)}, \cdots, \vec{z}_{(t,1)}, \ldots, \vec{z}_{(t,m)}, \cdots$$

$\vec{z}_{(i,j)}$ denotes the jth example of batch i. For each batch i, the data is independently identically distributed with respect to a distribution $\Pr_i(\vec{x}, y)$. Depending on the amount and type of concept drift, the example distribution $\Pr_i(\vec{x}, y)$ and $\Pr_{i+1}(\vec{x}, y)$ between batches will differ. The goal of the learner $\mathcal{L}$ is to sequentially predict the labels of the next batch. For example, after batch t the learner can use any subset of the training examples from batches 1 to t to predict the labels of batch $t+1$. The learner aims to minimize the cumulated number of prediction errors.

In machine learning, changing concepts are often handled by time windows of fixed or adaptive size on the training data [23, 33, 18] or by weighting data or parts of the hypothesis according to their age and/or utility for the classification task [29]. The latter approach of weighting examples has already been used for information filtering in incremental relevance feedback approaches [2, 3]. Here the earlier approach maintaining a window of adaptive size is explored. For windows of fixed size, the choice of a "good" window size is a compromise between fast adaptability (small window) and good generalization in phases without concept change (large window). The basic idea of adaptive window management is to adjust the window size to the current extent of concept drift.

The task of learning drifting or time-varying concepts has also been studied in computational learning theory. Learning a changing concept is infeasible if no restrictions are imposed on the type of admissible concept changes[3], but drifting concepts are provably efficiently learnable (at least for certain concept classes) if the rate or the extent of drift is limited in particular ways.

[3]E.g., a function randomly jumping between the values one and zero cannot be predicted by any learner with more than 50 percent accuracy.

Helmbold and Long [8] assume a possibly permanent but slow concept drift and define the extent of drift as the probability that two subsequent concepts disagree on a randomly drawn example. Their results include an upper bound for the extend of drift maximally tolerable by any learner and algorithms that can learn concepts that do not drift more than a certain constant extent of drift. Furthermore, they show that it is sufficient for a learner to see a fixed number of the most recent examples. Hence a window of a certain minimal fixed size allows to learn concepts for which the extent of drift is appropriately limited. While Helmbold and Long restrict the extend of drift, [20] determine a maximal rate of drift that is acceptable by any learner, i.e., a maximally acceptable frequency of concept changes, which implies a lower bound for the size of a fixed window for a time-varying concept to be learnable, which is similar to the lower bound of Helmbold and Long.

In practice, however, it usually cannot be guaranteed that the application at hand obeys these restrictions, e.g., a reader of electronic news may change interests almost arbitrarily often and radically. Furthermore the large time window sizes, for which the theoretical results hold, would be impractical. Hence more application-oriented approaches rely on far smaller windows of fixed size or on window adjustment heuristics that allow far smaller window sizes and usually perform better than fixed and/or larger windows [18, 33]. While these heuristics are intuitive and work well in their particular application domain, they usually require tuning their parameters, are often not transferable to other domains, and lack a proper theoretical foundation.

3.1 Window Adjustment by Optimizing Performance

Our approach to handling drift in the distribution of examples uses a window on the training data and SVM (Section 2, [32]) as its core learning algorithm. This window should include only those examples which are sufficiently close to the current target concept. Assuming the amount of drift increases with time, the window includes the last n training examples. Previous approaches used similar windowing strategies. Their shortcomings are that they either fix the window size [23] or involve complicated heuristics [18, 33]. A fixed window size makes strong assumptions about how quickly the concept changes. While heuristics can adapt to different speed and amount of drift, they involve many parameters that are difficult to tune. Here, we present an approach to selecting an appropriate window size that does not involve complicated parameterization. This is the first approach that is neither restricted to static concepts nor requires an extensive parameterization and that effectively and efficiently learns drifting concepts. They key idea is to select the window size so that the estimated generalization error on new examples is minimized. To get an estimate of the generalization error we use a special form of $\xi\alpha$-estimates [11]. $\xi\alpha$-estimates are a particularly efficient method for estimating the performance of a SVM. They can be computed after only a single SVM run on the training data. It is proven in [11] that the $\xi\alpha$-estimator of the error rate is an approximate upper bound on the number of leave-one-out errors in the training set, and it is shown that the estimator is pessimistically biased, overestimating the true error rate on average. Experiments show that the bias is acceptably small for text classification problems [11].

The window adjustment algorithm proposed here uses $\xi\alpha$-estimates in a particular way. At batch t, it essentially tries various window sizes, training a SVM for each resulting training set.

$$\begin{array}{r} \vec{z}_{(t,1)},\ldots,\vec{z}_{(t,m)} \\ \vec{z}_{(t-1,1)},\ldots,\vec{z}_{(t-1,m)},\vec{z}_{(t,1)},\ldots,\vec{z}_{(t,m)} \\ \vec{z}_{(t-2,1)},\ldots,\vec{z}_{(t-2,m)},\vec{z}_{(t-1,1)},\ldots,\vec{z}_{(t-1,m)},\vec{z}_{(t,1)},\ldots,\vec{z}_{(t,m)} \\ \vdots \end{array}$$

For each window size, it computes a $\xi\alpha$-estimate based on the result of training. The $\xi\alpha$-estimator used here considers only the last batch, that is, the m most recent training examples $\vec{z}_{(t,1)},\ldots,\vec{z}_{(t,m)}$. This reflects the assumption that the most recent examples are most similar to the new examples in batch $t+1$. The window adjustment algorithm selects the window size that minimizes the $\xi\alpha$-estimate of the error rate.

3.2 Experiments

Each of the following four data management approaches is evaluated in combination with the SVM. In the *full memory* approach, the learner generates its classification model from all previously seen examples, i.e., it cannot "forget" old examples. With *no memory*, the learner always induces its hypothesis only from the most recent batch. This corresponds to using a window of the fixed size of one batch. A window of the *fixed size* of three batches is also used. Finally, for a window of *adaptive size*, the window adjustment algorithm

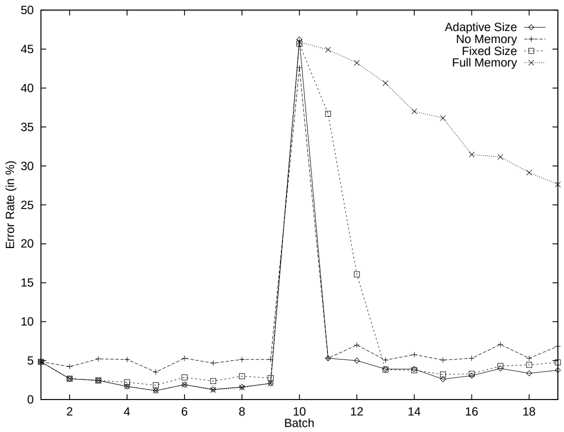

Figure 1 Comparison of the prediction error rates for one of the three scenarios (scenario A, averaged over 10 runs). The x-axis denotes the batch number and the y-axis the average prediction error.

proposed in the previous section adapts the window size to the current concept drift situation.

The experiments are performed in an information filtering domain. Text documents are represented as attribute-value vectors (bag of words model), where each distinct word corresponds to a feature whose value is the TF/IDF-weight of that word in that document. Words occurring less than three times in the training data or occurring in a given list of stop words are not considered. Each document vector is normalized to unit length to abstract from different document lengths. The experiments use a subset of 2608 documents of the data set of the Text REtrieval Conference (TREC) consisting of English business news texts. Each text is assigned to one or several categories, five of which are considered here. For the experiments, three concept change scenarios with different types of concept drifts are simulated (see [17] for more details). The texts are randomly split into 20 batches of equal size containing 130 documents each (and hence leaving eight randomly remaining documents unused in each trial). The target concept, i.e. the correct label of a document at a certain point in time (batch), reflects whether that document is (currently) relevant to the current user interest.

Table 1 shows the results of the different memory management approaches on the three simulated scenarios averaged over ten runs and over the 20 batches of each run. Figure 1 compares the prediction error rate of the adaptive window size algorithm with the non-adaptive methods (for scenario A). In all three scenarios, the full memory

Table 1 Prediction error of all window management approaches for three scenarios averaged over 10 trials with 20 batches each (standard sample error in parentheses)

Scenario	Full Memory	No Memory	Fixed Size	Adaptive Size
A:	20.36%	7.30%	7.96%	5.32%
	(4.21%)	(1.97%)	(2.80%)	(2.29%)
B:	20.25%	9.08%	8.44%	7.56%
	(3.56%)	(1.57%)	(2.00%)	(1.89%)
C:	7.74%	8.97%	10.17%	7.07%
	(3.05%)	(2.84%)	(3.30%)	(3.16%)

strategy and the adaptive window size algorithm essentially coincide as long as there is no concept drift. During this stable phase, both show lower prediction error than the fixed size and the no memory approach. At the point of concept drift, the performance of all methods deteriorates. While the performance of no memory and adaptive size recovers quickly after the concept drift, the error rate of the full memory approach remains high. Like before the concept drift, the no memory and the fixed size strategies exhibit higher error rates than the adaptive window algorithm in the stable phase after the concept drift. This shows that only the adaptive window size algorithm can achieve a relatively low error rate over all phases in all scenarios. This is also reflected in the average error rates over all batches given in Table 1. The adaptive window size algorithm achieves a low average error rate on all three scenarios. Similarly, precision and recall are consistently high [17]. A more detailed discussion of the results along with additional figures illustrating the performance of the different approaches in the three scenarios and the automatically chosen window sizes as well as results for the performance metrics recall and precision can be found in [17].

Summarizing this section, we proposed a new method for handling concept drift with support vector machines. The method directly implements the goal of discarding irrelevant data with the aim of minimizing generalization error. Exploiting the special properties of SVM, we adapted $\xi\alpha$-estimates to the window size selection problem. Unlike for the conventional heuristic approaches, this gives the new method a clear and simple theoretical motivation. Furthermore, the new method is easier to use in practical applications, since it involves less parameters than complicated heuristics. Experiments in an information filtering domain show that the new algorithm achieves a low error rate and selects appropriate window sizes over very different concept drift scenarios.

4 Transduction: Learning from Labeled and Unlabeled Data

The idea of transduction was introduced by Vapnik (see Chapter 10 in [31]). Often it is not necessary to learn a general classification rule for arbitrary test examples. Instead, the easier problem of only classifying a specific already known test set with as few errors as possible has to be solved. First experiments Joachims conducted in a text classification domain showed that the minimization of the probability of an error on a specific test set leads to better results than learning a general classification rule [10, 12]. Obviously the unlabeled test examples provide additional information for a transductive learner not only using the labeled training examples, information about the distribution of the test data that the learner can leverage on. This section investigates when and why transduction works better than the standard induction approach. After reviewing transductive support vector machines and providing a positive and a negative case study, we discuss properties of a domain that enable transduction to be helpful.

4.1 Transductive Support Vector Machines (TSVM)

Inductive support vector machines have already been described in Section 2 along with the primal and dual optimization problems 1 and 2 they solve. The description of transductive inference here follows the presentation in [10]. For a learning task $P(\vec{x}, y) = P(y|\vec{x})P(\vec{x})$ the learner $\mathcal{L}$ is given a hypothesis space H of functions $h : X \longrightarrow \{-1, 1\}$ and an i.i.d. sample S_{train} of n training examples

$$(\vec{x}_1, y_1), (\vec{x}_2, y_2), ..., (\vec{x}_n, y_n).$$

Each training example consists of a vector $\vec{x} \in X$ and a binary label $y \in \{-1, +1\}$. In contrast to the inductive setting, the learner is also given an i.i.d. sample S_{test} of k test examples

$$\vec{x}_1^*, \vec{x}_2^*, ..., \vec{x}_k^*$$

from the same distribution. The transductive learner $\mathcal{L}$ aims to select a function $h_{\mathcal{L}} = \mathcal{L}(S_{train}, S_{test})$ from H using S_{train} and S_{test} so that the expected number of erroneous predictions

$$R(\mathcal{L}) = \int \frac{1}{k} \sum_{i=1}^{k} \Theta(h_{\mathcal{L}}(\vec{x}_i^*), y_i^*) dP(\vec{x}_1^*, y_1^*) ... dP(\vec{x}_k^*, y_k^*)$$

on the test examples is minimized. $\Theta(a, b)$ is zero if $a = b$, otherwise it is one. Solving this optimization problem means finding a labeling $y_1^*, ..., y_k^*$ of the test data and a hyperplane $< \vec{w}, b >$, so that this hyperplane separates both training and test data with maximum margin. To be able to handle non-separable data, we can introduce slack variables ξ_i similarly to the way we do with inductive SVM.

Optimization Problem 3 (Transductive SVM)

Minimize over $(y_1^*, ..., y_k^*, \vec{w}, b, \xi_1, ..., \xi_n, \xi_1^*, ..., \xi_k^*)$:

$$\frac{1}{2}||\vec{w}||^2 + C \sum_{i=0}^{n} \xi_i + C^* \sum_{j=0}^{k} \xi_j^*$$

$$\begin{aligned} \textit{subject to:} \quad & \forall_{i=1}^{n} : y_i[\vec{w} \cdot \vec{x}_i + b] \geq 1 - \xi_i \\ & \forall_{j=1}^{k} : y_j^*[\vec{w} \cdot \vec{x}_j^* + b] \geq 1 - \xi_j^* \\ & \forall_{i=1}^{n} : \xi_i \geq 0 \\ & \forall_{j=1}^{k} : \xi_j^* \geq 0 \end{aligned}$$

C and C^* are parameters set by the user. They allow trading off margin size against misclassifying training examples or excluding test examples. Joachims developed an algorithm to solve this optimization problem efficiently [10, 12]. Here SVM^{light} is also used for computing the solution of this optimization problem (see also Section 2).

4.2 Transduction for Text Classification and for Drifting Concepts

Empirical results show that unlabeled data can help to significantly improve the performance of text classifiers [10, 21, 24]. Joachims showed that especially in case of few labeled examples and many unlabeled examples, the use of unlabeled data can improve the classification performance significantly [10]. The transductive use of unlabeled data also lets the performance drop more gracefully if the number of labeled examples is reduced. As pointed out in [10, 12], it is well-known in information retrieval that words in natural language occur in strong co-occurrence patterns [30]. While some words are likely to occur in one document, others are not. This type of information is independent of the document labels and can be exploited, if unlabeled data is used.[4] In addition to

[4]For example the words *pepper* and *salt* as well as the words *atom* and *physics* have a high co-occurrence, while e.g., *pepper* and *atom* only rarely co-occur. This is for example helpful, if cooking recipes have to be separated from

the TSVM described above, there are other semi-supervised methods for exploiting unlabeled data (see [10] or [14] for a short overview).

In Section 3 information filtering was named as a typical application, where the target concept in a classification task, in this case the user interest, may change over time, which makes the learning task more difficult because the system should adapt to such changes. Another difficulty in this example domain is the fact that users often give little feedback and expect a filtering system to achieve a good performance, even if only a few labeled training examples are provided.

This section proposes an extension of the method for recognizing and handling concept changes with support vector machines (Section 3 and [17]). The extended method uses unlabeled data to reduce the need for labeled data [14, 16]. Its basic idea is to first use the algorithm described in [17] to find the right window size on the labeled training data, $win_{labeled}$, using $\xi\alpha$-estimates for an inductive SVM. Then an almost identical algorithm is used to determine a good window size on the unlabeled data, $win_{unlabeled}$, on the same stream of documents using $\xi\alpha$-estimates for a transductive SVM to estimate the prediction error on the test set, leaving the window size $win_{labeled}$ unchanged.

Why maintain separate window sizes $win_{labeled}$ and $win_{unlabeled}$ for labeled and unlabeled data? The probability $P(y|\vec{x})$ describing the user interest, i.e., the drifting concept captured by the labeled data, may change at a rate other than the probability $P(\vec{x})$, which describes the distribution of documents identically underlying both the labeled and unlabeled examples independent of their labels. Hence it is sensible to use separate windows to obtain the best information from both probability distributions.

The window adjustment algorithm for the labeled data using an inductive SVM and the training data only has already been described in Section 3. The modified window adjustment algorithm for the unlabeled data [14] uses a transductive SVM. According to the transductive setting, the test set, i.e., the examples in the batch $t+1$ are used as unlabeled examples in the optimization for learning a TSVM. For text classification tasks with stable, i.e., non-drifting concepts, the performance improvement of a TSVM as compared to an inductive SVM is maximal for small sets of labeled training examples and large sets of unlabeled examples used in addition [10]. Therefore not only the relatively small test set is considered useful unlabeled data, but also the unlabeled data in the current time window of size $win_{labeled}$, i.e., in the currently used part of the training set, and all training examples outside this time window, which are all treated as unlabeled examples, are considered potentially useful. First experimental results with this approach are encouraging [16].

4.3 Comparison of Inductive and Transductive Learning in a Semiconductor Rare Fault Detection Domain

To evaluate the use of transduction in a completely different application domain, inductive and transductive support vector machines (SVM and TSVM, respectively) were applied to a classification task from electrical engineering, namely a semiconductor rare fault detection task previously used as a benchmark for computational intelligence methods within the DFG Collaborative Research Center on Computational Intelligence (SFB 531). The data consist of bond resistance values measured in stress tests with 34 high performance transistors until these transistors failed. Given the bond resistance values of such a transistor of the last four time steps, the task is to predict whether or not the transistor will fail within the next seven time steps. The 34 time series of measurements are split into 29 series for training and 5 time series for testing resulting in 1157 training and 204 test examples for the time window of size four time steps described above.

For evaluating whether transduction can help to improve the performance of a support vector machine in this domain, we trained SVM and TSVM with linear, RBF, and polynomial kernel on the training data and compared their performance on the test set. The TSVM was allowed to use the complete test set (without its labels of course) in its training. The experiments were performed using SVM^{light} [9, 12]. Table 2 shows the best results obtained by SVM and TSVM with these kernels trying a few parameter variations (see [4] for more details on the parameterization and the results). While for the linear kernel, the SVM seems to benefit from transduction, it does not for the other two kernels, i.e., RBF and polynomial kernel. However, the differences in performance cannot be considered statistically significant, since the data set is used, exactly as for the previously tested methods, on a fixed single split of the data set into training and test data only. In contrast to

texts about atom physics, and the training documents contain only some of these words, while the test documents contain some of the others along with the corresponding co-occurrences.

Table 2 Error of SVM and TSVM with different kernels on the semiconductor fault detection task

Error	SVM, linear kernel	SVM, RBF kernel	SVM, polynom kernel
Induction:	10.80 %	9.20 %	9.20 %
Transduction:	9.60 %	10.00 %	10.00 %

the positive results in the text classification domain, there seems to be not much potential for transduction on this failure detection data set.

4.4 Conclusions: When Does Transduction Work Well?

While transduction leads to performance improvements in text classification tasks, where unlabeled data can be used to lower error rates or to reduce the need for labeled data, it does not seem to be very helpful in the second case study of semiconductor failure detection. So the questions arises, what the difference between these two domains is, and when transduction can be expected to be helpful, i.e., which properties a domain should have to benefit from transduction.

There are two intuitive reasons why transduction works well in text classification domains. First, as explained above, texts of similar or identical topics show strong word co-occurrence patterns in a sparse feature space. These co-occurrence patterns often indicate a topic membership before the topic label of a document is known. Hence unlabeled examples can be used to better estimate the boundaries of a topic than is possible from fewer labeled examples alone. Second, as shown in [10, 16], a learner particularly benefits from transduction, if many unlabeled examples and only few labeled examples are available. In most text classification domains, there are huge numbers of unlabeled documents available, which usually by far exceed the number of available labeled documents. So in text classification tasks, transduction can leverage from these two aspects.

The second domain, the rare fault detection task, exhibits neither of these two properties. First, the number of test examples is rather small compared to the number of training examples. Second, the feature space is not sparse, but very dense, and it is not clear whether there are any co-occurrence patterns within the two classes "will fail (within seven time steps)" and "will not fail". Summarizing this discussion, good prerequisites for transduction to work well seem to be strong co-occurrence patterns between examples of the same class and a large number of test examples compared to the number of training examples.

5 Learning with Prior Knowledge

In machine learning one can distinguish two kinds of knowledge, namely background knowledge and prior knowledge. In the case of learning with prior knowledge, the underlying learning task can be eased by introducing additional, domain-specific knowledge. This knowledge restricts the hypothesis space and thus allows faster and/or better learning. Background knowledge extends the examples and thus the search space and therefore makes the learning task more difficult.

One classical setting for the learning task learning with background knowledge is the following: The learned hypothesis H has to be consistent not only with the given example set E, but also with the given background knowledge T, and any positive but no negative example should be deducible from the union of hypothesis h and background knowledge T. This inclusion of background knowledge complicates the given learning task [25], and restrictions of the representation formalism are necessary to make learning possible in polynomial time [13]. The inclusion of background knowledge makes learning more difficult, because on the one hand, more complex hypotheses can be built. On the other hand, the integration of new knowledge becomes harder the more knowledge is given, since the number of possible inconsistencies increases.

In contrast to this kind of knowledge that enlarges the hypothesis space, the use of prior knowledge permits scaling down the hypothesis space and/or controlling the search in the search space. Restricting the hypothesis space adequately may accelerate learning in terms of running time of the algorithm and/or improve the learning performance, e.g., in terms of classification/regression error or cost on the test examples. This section presents an approach to SVM making use of prior knowledge about the importance of particular examples in an example set and appropriately adapting the SVM optimization problem.

5.1 Using Additional User Knowledge in Support Vector Machines

Many applications of machine learning involve the learning of classifiers. Given a set of labeled training examples, the task is to learn a classifier for predicting the labels of previously unseen examples. By providing the labels of the training examples, the users already specify a lot of their knowledge about the classification problem at hand. In some cases, however, the users may not be satisfied with the result provided by the learning method. Hence the users may want to specificy additional knowledge about the problem or constraints on the desired solution and they may want the learner to provide a classifier that better fits their needs. The goal of this research is to allow the users to specify additional knowledge about the classification problem and to incorporate this knowledge into the learning process. In [15] support vector machines [32] (Section 2) were chosen as learning methods for classifiers, and weights for individual training examples provided by the user are considered in the training of SVM.

User knowledge about a domain can often be expressed by weights for the examples provided to the learner. Consider for example a medical domain, where it is essential to classify critical cases correctly, because a person's life may depend on it, while incorrect classifications of non-critical cases may be more acceptable. Similarly, one can use weights to enforce the proper classification of typical cases of a disease and to put less emphasis on non-typical cases, i.e., cases where even the expert is not sure or where the appearance of a disease is not clearly recognizable. Weights can also be used to express the confidence one has in the correctness of the classification of the examples, e.g., to express the reliability a certain example source.

We extended the formulation of the primal SVM optimization problem (optimization problem 1 in Section 2) for weighted examples and derived the corresponding dual SVM optimization problem [15]. For each training example $\vec{x}_i \in \mathbb{R}^{\mathbb{N}}$, the user provides a label $y_i \in \{-1, +1\}$, indicating its classification and a weight $C_i \in [0, 1]$ (or alternatively $C_i \in \mathbb{R}^+$) indicating the importance of the correct prediction of the class label of this example $(i = 1, ..., n)$.

Optimization Problem 4 (SVM (with weights))

$$
\begin{array}{ll}
\textit{Minimize:} & V(\vec{w}, b, \vec{\xi}) = \frac{1}{2}\vec{w} \cdot \vec{w} + \sum_{i=1}^{n} C_i \cdot \xi_i \\
\textit{subject to:} & \forall_{i=1}^{n} : y_i[\vec{w} \cdot \vec{x}_i + b] \geq 1 - \xi_i \\
 & \forall_{i=1}^{n} : \xi_i \geq 0
\end{array}
$$

6 Feature Selection and Generation

The non-standard learning tasks that are considered in this section are *feature selection* and *feature generation*. They can be viewed as two sides of the representation problem, i.e., the problem of finding an adequate representation language L_E for the learning task at hand. In cases where this language in terms of the original feature set is not sufficient to describe the problem, feature generation helps to enrich the language by constructing additional features. In cases where the representation language contains more features than necessary, subset selection helps to simplify the language [5]. In this section, we present a framework that solves the representation problem by connecting feature generation and selection in a combined evolutionary approach.

6.1 Feature Generation Using Type-Restricted Generators

Feature generation can be seen as the process of creating features by applying feature constructors, taking the original features as input and producing a modified feature set containing additional constructed features. The main motivation for the use of feature generation is to create hypotheses that were not representable in the original representation language [22].

We propose a general framework for the generation of features, subdividing the generation process into the following main steps: Given a set of features $F = f_1, .., f_n$ and feature generators $G = g_1, ..g_k$, we first choose a particular feature generator $g_j \in G$ for the generation process. Then, by checking the types of all features in F, the compatible feature subsets $F_c = F_{c_1}, .., F_{c_l} \subseteq F$ are determined with regard to the type restrictions of the generator at hand.[5] Finally, the feature generator g_j is applied to the set of compatible features (or a subset thereof) and the resulting features are added to the original feature set F. The features $f_i \in F_{c_i}$ are not restricted to original features but can already be compound features that have previously been created by a generator. This generation concept allows a recursive and thus arbitrarily complex feature generation. The set of applicable feature constructors may include general functions, such as basic mathematical operations, as well as domain-specific generators. An obvious advantage of using type-restricted constructors is the reduction of the set of constructible features to a well-formed subset and thus the restriction of the search space extension to useful subspaces. The presented generator concept is an integral part of the following approach that combines feature generation and selection in a common framework.

[5] The concept of meta-knowledge in terms of feature types is described in more detail in Section 7.

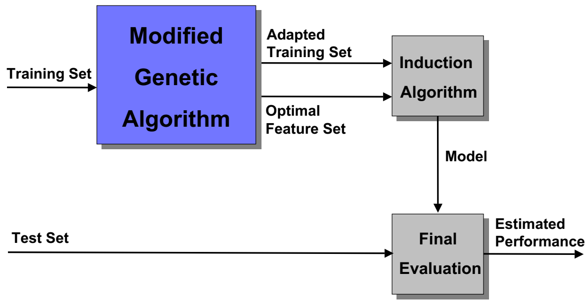

Figure 2 Combined wrapper-based feature selection and generation approach

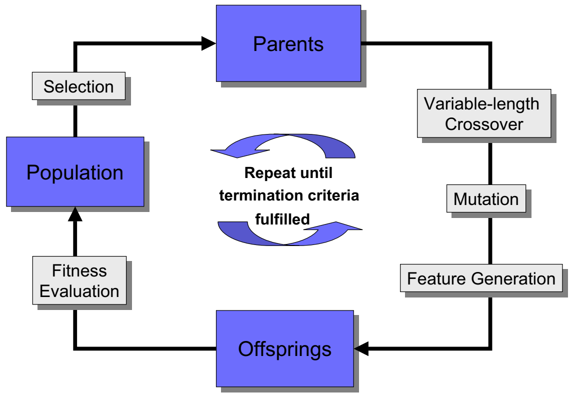

Figure 3 The modified genetic algorithm

6.2 Combined Approach to Feature Generation and Selection

In this section we introduce a wrapper-based approach [19], using a modified genetic algorithm that incrementally selects and generates new features and an induction algorithm as learning and evaluation method. The general idea is to deploy the positive search properties of conventional genetic algorithms for the incremental adaptation of a given feature space. Genetic algorithms have proved to work well on feature selection problems, where the search space produced by the initial feature set already contains the searched hypothesis (see e.g. [34]). In cases where this premise is not fulfilled, we have to create new features to adequately extend the search space, which can be done by applying the given feature generators to feature subsets with compatible feature types.

According to our approach [26], the feature transformation process, which is done by means of a modified genetic algorithm, is wrapped around an arbitrary induction algorithm. Thus, the genetic algorithm conducts the search for a good feature subset using the induction algorithm for the evaluation of the current feature subsets. The induction algorithm is run on a data set, which is usually partitioned into internal training and hold-out sets, with different sets of features removed from and added to the data accordingly. The process of generating feature sets, using the modified genetic algorithm, and evaluating these sets is repeated until a given termination criterion is fulfilled. The resulting feature set is chosen as the final set on which to run the induction algorithm. The final evaluation of the resulting classifier is done using an independent test set that was not used during the search (see Figure 2).

The key idea in using the incremental feature generator in the context of a genetic algorithm is that any feature can be generated with a probability $P > 0$ in a finite number of iterations, i.e., generations, given a particular generator and the original features. Thus, more formally, given a feature set $\vec{f_t}$ at time-point t, a generator G with $G(\vec{f_t}) = \vec{f}_{t+1}$ and f being the searched feature, the following proposition can be assumed:

$$f \in \lim_{t \to \infty} G(\vec{f_t})$$

The basis for our combined approach to feature generation and selection is a canonical genetic algorithm, as e.g. described in [7]. It consists of an n-tuple of binary strings b_i of length l, where the bits of each string are considered to represent single features and where the n-tuple represents a feature set. Following the terminology of biologic evolution the operations performed on the population are called *mutation*, *crossover*, and *selection*. Each individual represents a feasible solution of a given problem and its objective function value $\phi(b_i)$ is said to be its *fitness*, which has to be maximized.

The task is to search for a good representation by selecting and generating features. We therefore partially adapted the standard genetic algorithm operators. Figure 3 shows the corresponding modified genetic algorithm. Here, crossover recombines different feature sets and mutation selects and accordingly deselects single features on a particular chromosome, i.e., feature set. In addition to the set of standard GA operators for feature selection, we applied the presented feature

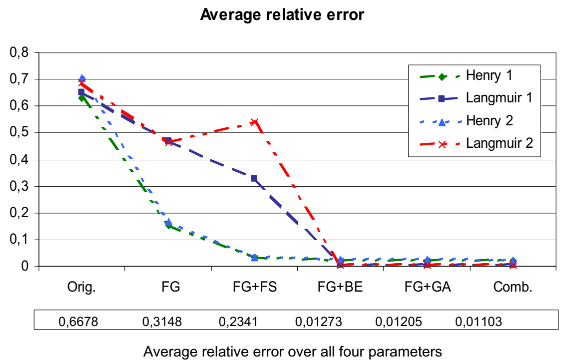

Figure 4 Average relative error for the target values Henry and Langmuir on a two-substance mixture, tested on different operator chains

generator. To ensure that the best individual of a generation remains in the population, we used an elitist selection strategy.

6.3 Experimental Evaluations

The combined approach was evaluated by an application problem from the field of chromatography. The learning task considered here was to predict two components, or more exactly, the four characteristic coefficients of a two-component mixture. Each of the two components was given in terms of a time series. Since the learning task was a regression problem, the learning method used throughout the following experiments is a regression SVM [28]. The learning performance was evaluated using relative error, comparing the predicted and the real values of the four coefficients (Henry and Langmuir constant for each substance). Based on the structure of the overall learning task, we systematically compared the performance of several learning chains, comprising different types of data preprocessing in terms of feature generation and selection. Figure 4 shows the experimental results.

In the first experiment we simply used the original (time series) features to learn and evaluate an SVM model without any preprocessing steps. Due to noise in the simulated measurements of the original features, this operator chain only yielded a poor prediction performance.

The second chain additionally contained a preprocessing operator that generated numeric characteristics from the original time series data. Since the sensor readings might be noisy and perhaps slightly shifted along the time axis between different measurements, the individual feature in the time series of an example, i.e., the concentration measured at one particular point of time, is not very reliable. Accordingly, the construction of these robust features significantly improved the learning result.

Since it was not obvious which of the new and which of the original features were really helpful in solving the learning task, a feature selection step was performed subsequent to the feature generation step to obtain a feature set well-suited to the given learning task. Different feature selection wrappers, namely forward selection (FS), backward elimination (BE) [1], and a genetic algorithm (GA) were employed, reducing the set of features and increasing the learning performance in terms of the prediction error.

Finally, we tested the combined approach on the given learning problem. In this setting, feature generation and selection were not used as subsequent preprocessing steps but intertwined in a feature wrapper approach. This approach yielded the best prediction performance of all tested learning chains.

7 Learning Environment YALE

As we have seen in the last sections, data often has to be preprocessed to be usable by a given machine learning method and to achieve an acceptable level of prediction performance. One of the central problems in this context is the choice of an adequate example representation by a good set of features. To handle this problem, it is often necessary to construct complex experiment chains, combining different preprocessing and learning steps, rather than using a single learning scheme. To efficiently study the described non-standard learning tasks we built the flexible, platform-independent learning environment YALE (Yet Another Learning Environment) [6, 27], which allows to easily specify and execute such experiment chains.

7.1 Existing Environments

There already exist several machine learning and data mining environments that provide a number of methods from machine learning, statistics, and pattern recognition. Two of the most popular non-commercial environments are WEKA[6] (Waikato Environment for Knowledge Analysis), developed at University of Waikato, NZ, and MLC++[7], first

[6] `http://www.cs.waikato.ac.nz/ml/weka/`
[7] `http://www.sgi.com/tech/mlc/`

developed at Stanford University and then extended by Silicon Graphics, Inc. (SGI).

WEKA is a collection of machine learning algorithms implemented in Java. It supports a large number of learning methods for classification and regression. WEKA offers some preprocessing algorithms for the manipulation of features as well as three basic feature selection schemes, namely the feature correlation based approach, the wrapper approach, and the filter approach. Additionally, WEKA provides meta classifiers like bagging and boosting. MLC++ is a library of C++ classes for supervised machine learning. It provides a number of learning schemes similar to those used in WEKA. Additionally, wrappers around these basic inducers such as a discretization filter, a bagging wrapper, and a feature selection wrapper are provided.

Unfortunately, none of these machine learning and data mining environments is suited to handle the considered non-standard learning tasks, because they both neither support the composition and analysis of complex operator chains consisting of different nested preprocessing, learning, and evaluation steps nor sophisticated feature generators for the introduction of new features. An additional shortcoming of WEKA is its lacking scalability. It expects the example set to fit completely into main memory, which is not feasible for many data mining tasks. Furthermore, it is very slow on large data sets. YALEoffers a wrapper to the numerous learners and clusterers provided by WEKA to offer the same variety of methods within complex operator chains without reimplementation.

7.2 General Concepts

Real-world learning tasks are often solved by a sequence or combination of several data preprocessing and machine learning methods. In YALE, each of these methods is regarded as an operator. A sequence of such operators is called an operator chain, wherat an operator chain again is an operator, both in the sense of a definition as well as in the object-oriented programming sense. Operators may enclose other operators or operator chains and are then often referred to as *wrappers*. Typical examples of wrappers are cross validation and feature selection wrappers. One central aspect is, that by enclosing other operators or operator chains, operators are arbitrarily nestable, so that even complex experimental setups can be built.

Operators require certain objects to be present in their input and guarantee others to be in their output. During its execution, an operator may modify, remove, or add objects before passing the objects to the next operator in the operator chain. YALE verifies that each operator receives its required inputs before executing an operator chain. Among the objects passed between operators in an operator chain are e.g. example sets, classification and regression models, and performance evaluation results. Operators are easily exchangeable by other operators in order to support the comparison of different operators and operator chains for the same subtasks. The only premise is that subsequent operators in an operator chain, or more generally all operators that share a common interface, have conformable input and output types.

YALE can process data sets that can be described in a single table, i.e., in an attribute-value vector format, in which each example is described by an attribute-value vector of equal fixed length. The scalability and applicability of the system is achieved by a sophisticated data handling concept. Data can be read from files, main memory, or (as a future option) from a database, whichever seems to be most appropriate for the current task. This can be done without making changes in the learning operators when varying the data source or switching between keeping all or just one example in main memory at a time.

While many learning systems do not consider additional knowledge for the learning process, YALE supports the optional use of meta-knowledge. Therefore it provides an additional ontology-based data structure specifying the meta-data of the given attributes. The information about the attribute types of an example is useful for feature generators, which can check their applicability on given attributes by verifying their attribute types.

YALE's usability to support elaborate studies of different non-standard learning tasks has successfully been demonstrated on several real-world applications (see e.g. Sections 3.2, 4, and 6.3).

8 Conclusion and Perspective

The assumptions that are often implicitly made when treating an application problem as a standard learning tasks, such as classification or regression, often do not hold in practical applications. The consequence of this inappropriateness is that the learning performance decreases or particular hypotheses cannot be learned. In this paper, several non-standard learning tasks including *learning drifting concepts*, *transduction*, *learning with prior knowledge*, and *feature generation and selection* have been studied, handling different kinds of inadequate problem representations.

For many real-world classification tasks, the concept to be learned may change over time and hence does not remain static as assumed in the standard learning task. Existing methods usually can only handle static concepts or require extensive, often domain-specific parameterization, to be adaptable to changing concepts and to achieve a satisfiable performance. In Section 3 we described and evaluated an effective and efficient method to learn such *drifting concepts* without extensive parameterization.

In other classification tasks, unlabeled data is available and can be used transductively to improve the performance of the classifier or to reduce the number of labeled examples needed to achieve an acceptable performance. We showed how unlabeled examples can be used to support the learning of drifting concepts and identified domain properties enabling transduction to lead to improved performance results. When a domain meets these properties, *transduction* can significantly reduce the need for labeled data and/or lead to lower prediction errors than a comparable inductive approach.

Sometimes the user wants to influence the learning result and provides additional knowledge about the domain. Section 5 showed a way how this may be done by extending SVM to use weighted examples.

In Section 6 we stated that, in cases where the original feature set is inadequate regarding the given learning task, one can significantly improve the learning performance by adding relevant features by means of *feature generation* and removing irrelevant features by applying *feature selection* methods. Furthermore, a framework was presented that applies feature generation and selection in a combined approach using an evolutionary based feature wrapper. This approach yielded an improved prediction performance compared to simple feature selection wrappers not using an additional feature generation component.

Section 7 proposed YALE, a learning environment supporting the study of non-standard learning tasks, especially the tasks mentioned above. YALE easily allows the execution and evaluation of complex nested chains of data preprocessing and machine learning operators.

One of the main future objectives is the consideration of learning tasks on a meta-level. The central question in this context is, in which form the configuration of learning tasks, i.e. the choice of a particular feature and example set, the concrete parameter settings, the choice of a specific learning method etc. can be automated.

9 Acknowledgements

This work was supported by the Deutsche Forschungsgemeinschaft (DFG), Collaborative Research Center on Computational Intelligence (SFB 531) at University of Dortmund.

References

1. D. W. Aha and R. L. Bankert. A comparative evaluation of sequential feature selection algorithms. In D. Fisher and H.-J. Lenz, editors, *Learning from Data*, chapter 4, pages 199–206. Springer, New York, 1996.
2. J. Allan. Incremental relevance feedback for information filtering. In H. P. Frei, editor, *Proc. 19th ACM Conf. on Research and Development in Information Retrieval*, pages 270–278. ACM Press, New York, 1996.
3. M. Balabanovic. An adaptive web page recommendation service. In W. L. Johnson, editor, *Proc. 1st Int'l Conf. on Autonomous Agents*, pages 378–385. ACM Press, New York, 1997.
4. G. Daniel, J. Dienstuhl, S. Engell, S. Felske, K. Goser, R. Klinkenberg, K. Morik, O. Ritthoff, and H. Schmidt-Traub. Novel learning tasks, optimization, and their application. In H.-P. Schwefel, I. Wegener, and K. Weinert, editors, *Advances in Computational Intelligence – Theory and Practice*, Natural Computing Series, chapter 8, pages 245–318. Springer, Berlin, 2002.
5. M. Dash and H. Liu. Feature selection for classification. *Int'l Journal of Intelligent Data Analysis*, 1(3):131–156, 1997.
6. S. Fischer, R. Klinkenberg, I. Mierswa, and O. Ritthoff. YALE: Yet Another Learning Environment - Tutorial. Technical Report CI-136/02, Collaborative Research Center 531, University of Dortmund, Germany, 2002.
7. D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Reading, MA, 1989.
8. D. P. Helmbold and P. M. Long. Tracking drifting concepts by minimizing disagreements. *Machine Learning*, 14(1):27–45, 1994.
9. T. Joachims. Making large-scale SVM learning practical. In B. Schölkopf, C. Burges, and A. Smola, editors, *Advances in Kernel Methods – Support Vector Learning*, chapter 11, pages 169–184. MIT Press, Cambridge, MA, 1999.
10. T. Joachims. Transductive inference for text classification using support vector machines. In *Proc. 16th Int'l Conf. on Machine Learning (ICML-99)*. Bled, Slovenia, 1999.
11. T. Joachims. Estimating the generalization performance of a SVM efficiently. In *Proc. 17th Int'l*

Conf. on Machine Learning (ICML-2000), pages 431–438. Morgan Kaufmann, San Francisco, 2000.

12. T. Joachims. The maximum-margin approach to learning text classifiers: methods, theory, and algorithms. PhD thesis, Artificial Intelligence Unit, Department of Computer Science, University of Dortmund, February 2001.
13. J.-U. Kietz. Induktive Analyse relationaler Daten. PhD thesis, Technical University of Berlin, October 1996.
14. R. Klinkenberg. Using labeled and unlabeled data to learn drifting concepts. In M. Kubat and K. Morik, editors, *Workshop notes of the IJCAI-01 Workshop on* Learning from Temporal and Spatial Data, pages 16–24. AAAI Press, Menlo Park, CA, 2001. Held at Int'l Joint Conf. on Artificial Intelligence (IJCAI).
15. R. Klinkenberg. Informed parameter setting for support vector machines: Using additional user knowledge in classification tasks. Technical Report CI-126/02, Collaborative Research Center 531, University of Dortmund, 2002.
16. R. Klinkenberg. Transductive learning of drifting concepts. Technical Report CI-125/02, Collaborative Research Center 531, University of Dortmund, 2002.
17. R. Klinkenberg and T. Joachims. Detecting concept drift with support vector machines. In *Proc. 17th Int'l Conf. on Machine Learning (ICML-2000)*, pages 487–494. Morgan Kaufmann, San Francisco, 2000.
18. R. Klinkenberg and I. Renz. Adaptive information filtering: Learning in the presence of concept drifts. In M. Sahami, M. Craven, T. Joachims, and A. McCallum, editors, *Workshop Notes of the Workshop* Learning for Text Categorization *held at the* 15th Int'l Conf. on Machine Learning (ICML-98), pages 33–40, AAAI Press, Menlo Park, CA, 1998.
19. R. Kohavi and G. H. John. Wrappers for feature subset selection. *Artificial Intelligence Journal, Special Issue on Relevance*, 97(1–2):273–324, 1997.
20. A. Kuh, T. Petsche, and R. L. Rivest. Learning time-varying concepts. In *Advances in Neural Information Processing Systems (NIPS)*, volume 3, pages 183–189. Morgan Kaufmann, San Mateo, CA, 1991.
21. C. Lanquillon. Partially supervised text classification: Combining labeled and unlabeled documents using an EM-like scheme. In R. López de Mántaras and E. Plaza, editors, *Proc. 11th Conf. on Machine Learning (ECML 2000)*, volume 1810 of LNCS, pages 229–237. Springer, Berlin, 2000.
22. C. Matheus. The need for constructive induction. In *Proc. 8th Int'l Workshop on Machine Learning*, pages 173–177. Morgan Kaufmann, San Mateo, CA, 1991.
23. T. Mitchell, R. Caruana, D. Freitag, J. McDermott, and D. Zabowski. Experience with a learning personal assistant. *Communications of the ACM*, 37(7):81–91, July 1994.
24. K. Nigam, A. McCallum, S. Thrun, and T. Mitchell. Text classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103–134, 2000.
25. G. D. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, chapter 8, pages 153–163. American Elsevier, New York, NY, 1970.
26. O. Ritthoff, R. Klinkenberg, S. Fischer, and I. Mierswa. A hybrid approach to feature selection and generation using an evolutionary algorithm. In John A. Bullinaria, editor, *U.K. Workshop on Computational Intelligence (UKCI 2002)*, pages 147–154. University of Birmingham, UK, 2002.
27. O. Ritthoff, R. Klinkenberg, S. Fischer, I. Mierswa, and S. Felske. YALE: Yet Another Machine Learning Environment. In R. Klinkenberg et al., editor, *LLWA 01 – Tagungsband der GI-Workshop-Woche* Lernen – Lehren – Wissen – Adaptivität, pages 84–92, Department of Computer Science, University of Dortmund, October 2001. *http://yale.cs.uni-dortmund.de/*.
28. S. Rüping. *mySVM-Manual*. Artificial Intelligence Unit, Department of Computer Science, University of Dortmund, 2000. *http://www-ai.cs.uni-dortmund.de/SOFTWARE/MYSVM/*.
29. C. Taylor, G. Nakhaeizadeh, and C. Lanquillon. Structural change and classification. In G. Nakhaeizadeh, I. Bruha, and C. Taylor, editors, *Workshop Notes on Dynamically Changing Domains – Theory Revision and Context Dependence Issues held at 9th European Conf. on Machine Learning (ECML-97)*, pages 67–78, 1997.
30. C. van Rijsbergen. A theoretical basis for the use of co-occurrence data in information retrieval. *Journal of Documentation*, 33(2):106–119, 1977.
31. V. N. Vapnik. *Estimation of Dependencies Based on Empirical Data*. Springer, New York, 1982.
32. V. N. Vapnik. *Statistical Learning Theory*. Wiley, Chichester, UK, 1998.
33. G. Widmer and M. Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(2):69–101, 1996.
34. J. Yang and V. Honavar. Feature subset selection using a genetic algorithm. In J. R. Koza et al., editor, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 380–385. Morgan Kaufmann, San Mateo, CA, 1997.