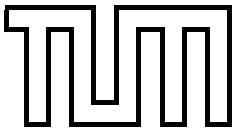
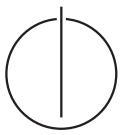


TECHNISCHE UNIVERSITÄT MÜNCHEN

FAKULTÄT FÜR INFORMATIK



Forschungs- und Lehreinheit VII:
Foundations of Software Reliability and Theoretical Computer Science

Siemens AG, Corporate Research and Technologies:
Information and Automation Technology, Intelligent Systems and Control

Ensemble Learning for Classification of Imbalanced Data

Ensemblebasierte Lernverfahren zur Klassifikation unbalancierter Daten

Master's Thesis in Informatik

Author:	Daniel Nyga
Supervisor:	Prof. Dr. Javier Esparza (TU München)
	Dr. Thomas A. Runkler (Siemens AG)
Advisor:	Dr. Clemens Otte (Siemens AG)
	Werner Hauptmann (Siemens AG)
Submission Date:	15.11.2010

Ich versichere, dass ich diese Master's Thesis selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

I assure the single handed composition of this master's thesis only supported by declared resources.

München, den 15.11.2010

Daniel Nyga

Abstract

This thesis investigates boosting algorithms for classifier learning in the presence of imbalanced classes and uneven misclassification costs. In particular, we address the well-known AdaBoost procedure and its extensions for coping with class imbalance, which typically has a negative impact on the classification accuracy regarding the minority class. We give an extensive survey of existing boosting methods for classification and enhancements for tackling the class imbalance problem, including cost-sensitive variants. Regularized boosting methods, which are favourable when dealing with noise and overlapping class distributions, are considered too. We theoretically analyze several strategies for introducing costs and their applicability in the case of imbalance. For one variant (AdaC1) we show that it is instable under certain conditions. We identify drawbacks of an often-cited cost-sensitive boosting algorithm (AdaCost), both theoretically and empirically. We also expose that an algorithm for tackling imbalance without using explicit costs (RareBoost) is a special case of the RealBoost algorithm, a probabilistic variant of AdaBoost. We approve our findings by empirical evaluation on several real-world data sets and academic benchmarks.

Zusammenfassung

In dieser Arbeit werden Boostingverfahren für das Lernen von Klassifikationsmodellen sowie deren Anwendung auf Probleme untersucht, die unbalancierte Klassen und ungleiche Fehlklassifikationskosten aufweisen. Im Besonderen werden das bekannte AdaBoost-Verfahren sowie dessen Erweiterungen zur Überwindung des Problems der Unbalanciertheit betrachtet, welches sich typischerweise negativ auf die Klassifikationsgüte bezüglich der unterrepräsentierten Klasse auswirkt. Die Einführung von Kostenparametern stellt hierfür einen beliebten Ansatz dar. Neben klassischen Boostingverfahren werden kostensensitive Verfahren ausführlich diskutiert. Regularisierte Boostingalgorithmen werden ebenfalls betrachtet, die vorzugsweise in Gegenwart von Rauschen und überlappenden Klassenverteilungen zum Einsatz kommen. Verschiedene Strategien werden theoretisch sowie empirisch in Bezug auf ihre Anwendbarkeit auf unbalancierte Probleme analysiert. Für ein Verfahren (AdaC1) wird gezeigt, dass es unter bestimmten Umständen instabil ist. Für einen weiteren vielzitierten kostensensitiven Ansatz (AdaCost) werden sowohl theoretische als auch praktische Nachteile aufgezeigt. Des Weiteren wird der RareBoost-Algorithmus, der auf explizite Kostenparameter verzichtet, als Spezialfall des RealBoost-Algorithmus identifiziert, einer probabilistischen Variante von AdaBoost. Die Ergebnisse dieser Arbeit werden empirisch anhand von Daten mehrerer realer Anwendungen sowie durch Benchmark-Daten bestätigt.

Contents

List of Figures

List of Tables

List of Algorithms

Notation

Symbol	Description
$\mathbb{R}$	Real numbers
N	Number of samples in a data set
$Z = (z_1, \ldots, z_N)$	Labeled data set including N samples
$z_n = (x_n, y_n)$	Sample with label y_n and feature vector x_n
X	Set of features
x	Feature vector
p	Dimensionality of a feature vector
y	Class label of a sample
$\mathscr{Y}$	Domain of a class label
$\mathscr{X}$	Domain of a feature
$f_m(x)$	Base classifier of iteration m
M	Number of base classifiers/iterations
m	Current iteration
$F(x)$	Combined ensemble output
$F_m(x)$	Combined ensemble output up to iteration m
α_m	Mixing coefficient of base classifier in iteration m
$w_m^{(n)}$	Weight of sample n in iteration m
$P(\cdot)$	Probability
$P(\cdot \mid \cdot)$	Conditional Probability
$P_w(\cdot \mid \cdot)$	Weighted conditional probability
$\mathbb{E}(\cdot)$	Expectation
$\mathbb{E}(\cdot \mid \cdot)$	Conditional expectation
$\mathbb{E}_w(\cdot \mid \cdot)$	Weighted conditional expectation
$\mathbf{1}_\pi$	Indicator function taking 1 iff π holds and 0 otherwise.
TP	True-Positive
FP	False-Positive
TN	True-Negative
FN	False-Negative

Chapter 1

Introduction

This chapter gives a brief overview of this thesis, introducing the concept of ensemble based systems and pointing out the challenges that may arise when dealing with imbalanced classes.

1.1. Ensemble Based Systems

What would you do?

Whenever humans have to make decisions of great importance they like to ask several other people for advice before making their final decision, in the hope that being aware of multiple different positions supports them in drawing a more informed conclusion. Before agreeing to a major medical procedure, for instance, we ask several doctors for their opinion, we read articles and reports of one's experiences before purchasing a big-ticket item and we delegate political liability preferably to a council than to a single person. In each case, the final decision what to do is obtained by a combination of the individual votes of several experts in order to retrieve the most promising selection in a set of alternatives, or to minimize the risk of an unfortunate one that has undesirable consequences.

In doing so, we pursue the primary goal of getting a preferably overall view on a particular issue without disregarding any important aspect. This requires the compound expertise of our consultants to indeed cover these aspects as completely as possible. Referring to the introductory examples above, in a political committee we would prefer to have at least one expert for various political matters instead of having a council consisting of equally skilled persons. Respectively, we want the doctors we are asking for advice to have different professional knowledge or experience.

The way how the individual choices are combined to reach a final decision can vary from case to case. On the political level, for example, the decision of the committee is often obtained by a unweighted majority vote. If we ask doctors for their opinion, one may have unquestionable confidence whereas in another one we trust only within limits, such that the former may have wider influence on our decision than the latter.

Although the strategy of "collecting several opinions" seems to be both quite effective and natural to humans, researchers in the computational intelligence community succeeded in adopting the idea of combining models in automated decision making not before the recent two decades. In this work we investigate ensemble based systems for classification, which are learning algorithms aspiring to build prediction models by combining the strengths of a collection of simpler base models [40], and their application to imbalanced or skewed data that typically have negative impact on a classifier's predictive performance.

Ensemble Learning in Automated Decision Making

In this section we briefly revisit the classification problem in general, we give an introduction to ensemble based classification systems, referring to two popular and well-known ensemble approaches, namely *bagging* and *boosting*. We motivate our investigations in boosting by discussing some of the most outstanding properties of each technique.

Classification Problems

Classification tasks are one of the central issues in the science of artificial intelligence and for a lot of industrial applications. In a typical classification scenario we wish to predict the class membership of a particular entity based on a set of observed properties, which are called *features*. Consider, for example, an OCR (Optical Character Recognition) application, which is given an optical representation of a distinct letter, and the system is supposed to infer which letter the person was intended to write down when she produced the document under consideration.

Typically, in such a scenario, we are given a *training set* $Z = \{z_1 = (x_1, y_1), \ldots, z_N = (x_N, y_N)\}$ of data, for which we know the outcomes or class labels y_n and the corresponding feature measurements x_n. Based on these data, we generate a *prediction model*, which enables us to assign a new feature vector $x = (x^{(1)}, \ldots, x^{(p)}) \in \mathcal{X}_1 \times \ldots \times \mathcal{X}_p$ to one of k classes $\mathcal{Y} \in \{\mathcal{Y}_1, \ldots, \mathcal{Y}_k\}$, where $\mathcal{X}_i$ denotes the domain

of the corresponding variable x_i. There are many other common terms, a prediction model for classification is referred to in literature, such as *classifier*, *hypothesis* or *learner*, which we use equivalently in this work. Furthermore, we will concentrate on binary classification problems (i.e. $k = 2$), where $y \in \{-1, +1\}$. Hence, more formally, we seek a function F that returns the corresponding class label y given an observation x,

$$F(x) : \mathcal{X}_1 \times \ldots \times \mathcal{X}_n \longrightarrow \{-1, +1\}, \tag{1.1}$$

which we want to learn from the set of examples in the training set.

Since the learning process is guided by an outcome variable y and we can assess the predictive performance of the learned model by inspecting the value y_i of each training sample x_i, a learning problem of that kind is referred to as a *supervised* learning task. By contrast, in an *unsupervised* learning task, we only deal with feature measurements and have no outcome variable y. In unsupervised learning, the goal is rather to learn how the input data are organized or clustered [40]. In this work, we will concentrate on supervised learning problems only.

Generalization, Overfitting and Regularization

Learning with the goal of making predictions differs from simple techniques such as memorization in the sense that the learned classifier is not only supposed to perform well on the training set of data, but also to achieve high classification accuracy when being faced with new, unseen data points.

A classifier's performance on data that have not been part of the model learning process is called the *generalization* or *test* performance. Thus, if we use all the data we have available for constructing a classifier, it is possible that the learning algorithm builds a classifier that perfectly predicts each point in the training set correctly, but fails on unseen data. In this case, the model is said to be *overtrained* or *overfitted*. Therefore, it is reasonable to split the data into two partitions, a *training set* which is used to build the classifier, and a *test set* for measuring how well the constructed model predicts unseen data.

There are a number of strategies how to split the data into test and training set, and how to use these for learning and evaluation. The technique that has been most approved by the machine learning community is *K-fold cross validation*.

In K-fold cross validation, we randomly divide the data into a partition of K subsets of equal size and use the union of $(K-1)$ of them for training the model and evaluate

its performance on the remaining one. This procedure is repeated K times, each time selecting a different subset to be the test set, such that we obtain K estimates of the predictive performance of our model, which are averaged to get a single final value.

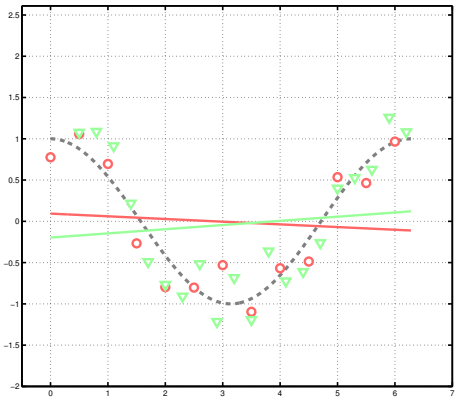
Assessing a classifier's performance is only one purpose cross validation is widely used for. In recent years, it has become a common practice to partition the data into three subsets, one training and one test set, and an additional *validation set*, which serves for pseudo testing [42]. Using a validation set, we continue training the classifier on the training set until we stop detecting an improvement in the predictive performance of the model applied to the validation set. When the improvement languishes, we stop the training process to prevent the model from overfitting. The same procedure can be employed for tweaking the parameters of a learning algorithm.

In order to illustrate the problem of overfitting, consider a problem where we have a target variable y and we are given a training data set X from which we want to learn the underlying distribution. Then, the expected generalization error $\mathrm{err}(F)$ over all data sets can be decomposed into [41]

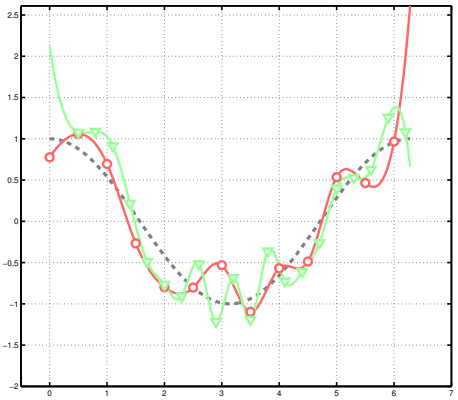
$$\mathrm{err}(F) = \underbrace{\mathbb{E}_Z\left(\left[F(x) - \mathbb{E}_Z(F(x))\right]^2\right)}_{\text{Variance}} \tag{1.2}$$

$$+ \underbrace{\left[\mathbb{E}_Z(F(x)) - \mathbb{E}_Z(y)\right]^2}_{\text{Bias}^2}, \tag{1.3}$$

where (1.2) and (1.3) are called *variance* and *bias*, respectively. Bias and variance are model-specific measures for the generalization error, where bias represents the deviation of the expected classifier output from the true class labels, and variance is defined as the expected deviation of a classifier's output from its expectation, both averaged over all data sets. In these terms, high bias means that the learned model does not pay enough heed to the data, i.e. the model is not sufficiently complex, whereas high variance indicates that there is the risk of overfitting. As an example, consider a simple regression problem as depicted in Figure 1.1. There are two sets of training data (drawn in red and green) which have been sampled from a cosine with additive gaussian noise. In Figure 1.1(a) we trained a linear model on each of the two training sets, whereas in Figure 1.1(b), we fitted two higher-order polynomials. As can be easily seen, the two linear models closely resemble, but their expected outputs are far from the true data, such they exhibit large bias. The polynomials fit their respective training sets perfectly, but their outputs highly depend on the data that have been used for training, such that they are characterized by high variance. None of the models is able to accurately fit the underlying cosine, plotted in dashed gray.



(a) *Linear Regression exhibiting large bias.*



(b) *Polynomial Regression exhibiting large variance.*

Figure 1.1.: *Example of prediction models with (a) high bias and (b) high variance.*

Ideally, we are seeking a model with both low bias and low variance, but in practice no model will perform equally well on all data sets, so there is the problem of finding a reasonable tradeoff between bias and variance. This procedure of optimizing the generalization behavior of a prediction model and mediating between these two extremes is called *regularization*.

Classifier Evaluation

It is important to know how well a classification system performs. Therefore, employing an appropriate evaluation measure is crucial for both assessing a classifier's performance and guiding classifier design. There are a number of characteristics, the compound of which give an impression of the classifier's performance. In this section we introduce a selection of the most important measures reported in literature.

Assume that we have a labeled data set Z available for testing, consisting of N tuples (x_i, y_i), specifying the input data and their corresponding class labels. The most natural way to conceive the error a classifier produces is to find the proportion of the number of misclassified samples

$$\varepsilon = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{F(x_i) \neq y_i}, \tag{1.4}$$

where $F(x)$ denotes the classifier output and $\mathbf{1}_\pi$ is the indicator function which evaluates to 1, if and only if the predicate π is true, and takes 0, otherwise.

		$F(x_i)$	
		-1	$+1$
y_i	-1	TN	FP
	$+1$	FN	TP

Figure 1.2.: *Generic confusion matrix for a binary classification problem: The outcomes of the predictions are spread out over the columns, the true class labels over the rows.*

Often it is desirable to know how the misclassification error is distributed among the classes. Hence, in order to get a more sophisticated view on the performance of a classifier, it is a common method to employ a *confusion matrix*, which is a $N \times N$ table holding the class-specific errors. In a confusion matrix, the value of each cell (i, j) is defined as

$$M_{i,j} = \sum_{k=1}^{N} \mathbf{1}_{F(x_k)=\mathscr{Y}_j \wedge y_k=\mathscr{Y}_i}, \tag{1.5}$$

such that the main diagonal holds the numbers of correctly classified instances. For a two class problem with $y \in \{-1, +1\}$, the four possible outcomes of a classification are named as the *true-positive* (TP), *true-negative* (TN), *false-positive* (FP) and *false-negative* (FN) cases, correspondingly to whether the prediction $F(x)$ is *positive* or *negative*, and its coincidence with the actual class label is *true* or *false*. Figure 1.2 shows a generic confusion matrix with the above definitions.

Based on the confusion matrix, there exist a number of measures revealing some more interesting properties of a classifier than (1.4) does, some of the most important shall be depicted in the following.

Traditionally, *accuracy* is the most commonly used measure in classifier design, which is nothing more than the residual of the error we defined in (1.4), in notation of the confusion matrix values,

$$a = \frac{TP + TN}{TP + TN + FP + FN}. \tag{1.6}$$

However, there are some shortcomings of this measure that may yield misleading results. Consider, for example, a two class problem where one class $\mathscr{Y}_1$ is represented by only 1% of the data. Then, the trivial classifier $F(x) \equiv \mathscr{Y}_2$, which constantly

True-Positive-Rate:	$TPR = \frac{TP}{TP+FN}$
True-Negative-Rate:	$TNR = \frac{TN}{TN+FP}$
False-Positive-Rate:	$FPR = \frac{FP}{FP+TN}$
False-Negative-Rate:	$FNR = \frac{FN}{FN+TP}$
Positive-Predictive-Value:	$PPV = \frac{TP}{TP+FP}$
Negative-Predictive-Value:	$NPV = \frac{TN}{TN+FN}$

Table 1.1.: *Error rates calculated using the confusion matrix.*

predicts the other class, may achieve accuracy of 99%, which seems to be very high, but is completely meaningless (this problem is discussed in detail in Section 1.2).

Instead of considering the overall misclassification error, it is more appropriate to differentiate between class labels. Table 1.1 shows a selection of error measures that give insight into a classifier's performance regarding the positive and negative predictions.

The *positive predictive value*, also called *precision*, denotes the proportion of all correctly predicted positive instances (with class label $y = +1$) and all instances that have been predicted to be positive:

$$p = \frac{TP}{TP + FP}. \tag{1.7}$$

In information retrieval, for instance, precision of classifier is the percentage of relevant documents that are identified for retrieval. Thus, precision can also be interpreted as a measure for the *correctness* of a classifier.

The *true positive rate*, defined as

$$r = \frac{TP}{TP + FN}, \tag{1.8}$$

is also known as *recall* and represents the percentage of all positive instances that actually could be identified by the learner, or, the percentage of retrieved documents that are relevant, in the information retrieval example. Hence, we can think of recall as a measure for the *completeness* of a classifier.

A more appropriate measure for the overall classification performance is given by the *F1-score* [43, 44], which takes the performance with reference to only one class into account and is defined as the harmonic mean of precision and recall,

$$F_1 = \frac{2 \cdot p \cdot r}{p + r} = \frac{2}{1/r + 1/p}. \tag{1.9}$$

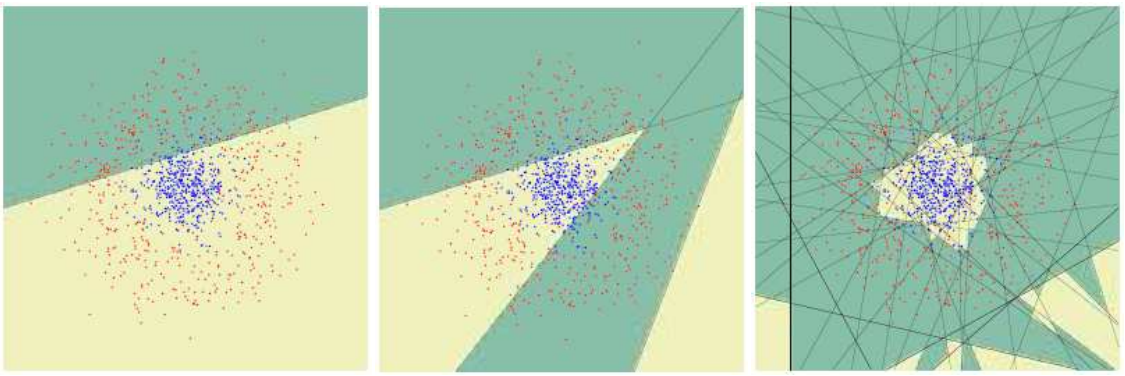
Both precision, recall and F-score reach their best valuation when they take 1 and their worst at 0. Since the harmonic mean of two numbers, in contrast to the arithmetic mean, tends to be closer to the smaller of the two, a high F-score indicates that *both* precision *and* recall are reasonably high.

Many kinds of classifiers, such as probabilistic models or neural networks, do not only return a binary outcome, but they assign a confidence score to their decision, e.g. $F(x) \in [-1, +1]$. Consider, for example, a model, where the sign of its output $\text{sgn}(F(x))$ specifies the predicted class label and the absolute value $|F(x)|$ can be regarded as the confidence in the prediction. Dealing with such models, the sensitivity of a trained model can be manipulated by varying the position of the decision threshold. Each threshold yields a pair of values (FPR, TPR) as defined above, which can be plotted in a graph. The result is a *receiver-operating-characteristics* (*ROC*) curve [74]. A model just making random guesses we would expect to reside along the diagonal connecting $(FPR = 0, TPR = 0)$ and $(FPR = 1, TPR = 1)$, whereas the ideal model achieves an FPR of 0 and a TPR of 1. ROC curves are a quite popular evaluation measure since they give a good overall picture of a classifier's performance. By computing the *area under the ROC curve* (*AUC*) we can transform the points of a ROC curve into a single value. However, as He and Garcia [5] point out, it is also possible for a high AUC classifier to perform worse in a specific region in ROC space [15, 74].

It has been shown by Davis and Goadrich [14] that there is a deep connection between the F-score measures and ROC analysis, and their expressiveness closely resembles. Since the former is used in most of the work we refer to in this thesis, we therefore pick up on this convention and employ precision, recall and F-score measures in our experiments.

Ensembles of Classifiers

As pointed out in the beginning of this chapter, humans often improve the quality of their decisions by taking several different opinions into account. In 1990, Hansen and Salamon [45] showed that the performance of neural networks can be improved by combining several similarly set up networks. Since then, huge efforts have been made by researchers in the computational intelligence community towards multiple



(a) $M = 1$ *(b)* $M = 2$ *(c)* $M = 40$

Figure 1.3.: *Example of an ensemble based system using linear discriminant functions as base classifiers: The individual classifiers perform only poorly as in (a) and (b), the ensemble of 40 base learners is able to separate the two classes with high accuracy (c). (Pictures taken from [75])*

classifier systems. In this section we give a brief overview on ensemble based techniques, outlining the cornerstones and most outstanding developments of the past.

In the machine learning literature an ensemble based system is referred to as a learning algorithm that strategically builds a set of models and makes predictions on a new, unseen data point by combining the individual decision of each of these models to one single decision by some combination rule [22].

An example of an ensemble of linear discriminant classifiers applied to a binary two-dimensional problem is given in Figure 1.3. Since the two classes are not linearly separable, the poor performance of the single classifier in Figure 1.3(a) is not surprising, but the classification accuracy can be improved by combining it with another similar (but not equal) base learner as shown in 1.3(b). Having trained 40 of them, the combined decision boundary separates the two classes quite well.

Dietterich [22], Kuncheva [42] and Polikar [23] give several views that motivate the use of an ensemble of classifiers instead of relying on a single one.

One reason why it is a good idea to employ an ensemble is *statistical*. Suppose we have a set of *different* classifiers that perform well on a given training set of data. If we are using classifiers having sufficient model complexity, it is even possible that all of them have a training error of zero, such that they are indistinguishable (with regard to the training error). However, they might have different generalization behavior. For classifying a new data point we are now supposed to pick one out of the set such that we are running into risk of selecting just an inadequate one that fails on the

new point. A safer option would be to, instead of picking one single classifier, charge all of them and somehow "average" the individual votes. The resulting model is not guaranteed to perform better than a single one, but this way we are able to diminish the risk of picking the wrong classifier.

A second reason given in [22, 42, 45] is *computational*. A lot of learning algorithms rely on numerical optimization techniques for optimizing a particular objective function, such as gradient descent, random search or genetic algorithms. As long as the objective function is a non-convex function, there is always the risk that optimization runs into a local optimum. Assuming that the learning algorithm of each classifier starts at different positions in the space of classifiers, we expect the learners to converge to different local optima. Training multiple classifiers there is a better chance to overcome these local optima, and combining the individual classifiers may lead to a more accurate decision.

A third argument is *representational*. It is often difficult to choose an appropriate space of classifiers to achieve good generalization performance. If the space is chosen too huge, there is the risk that the classifier overfits. If it is restricted too much, then the optimal classifier we are seeking will not belong to this space. Consider again the example shown in Figure 1.3, where the optimal classifier is not in the set of linear discriminant functions, but the ensemble yields a good approximation.

Summarizing, as Hansen and Salamon [45] point out, traditional standard practice in machine learning dictates to perform some trial tunings to a model, e.g. by means of cross validation as described above, to find an acceptable model and then to trust all future classifications to the best model found. However, as ensemble techniques suggest, it is more preferable to keep the complete set of models and run them all with an appropriate collective strategy. Following Polikar [23] and Friedman *et al.* [40], we define an ensemble of classifiers as follows:

Definition: Ensemble Based Classifier

> An ensemble of classifiers consists of a strategy for generating a diverse set of classifiers and a strategy for combining them to reach a final decision. The individual classifiers are called base classifiers.

As this definition and our introductory real-world examples of committees make apparent, there are two key questions that need to be answered when designing an ensemble system:

1. How will individual classifiers be generated?
2. How will the classifiers be combined to get the final decision?

In the following two sections we will briefly go into some essential aspects of each of these two components.

Creating an Ensemble

The strategy for creating a set of base learners ultimately affects the overall performance of the ensemble. The key idea of ensemble based systems is to build many classifiers, such that the committee of base learners is able to compensate for wrong decisions of individual ones, and that the combination results in improved generalization performance. However, this requires that the single base learners fail on different instances and thus we want the single classifiers to be as unique as possible. Such a set of classifiers is said to be *diverse* [23].

There are several ways of how to obtain an ensemble of diverse classifiers. Figure 1.4 shows a schematic representation of how an ensemble works and highlights different levels of where to approach for ensuring diversity among base learners [42]. Level A, *Combination Level*, will be the topic of the subsequent section.

The ensemble learning algorithms considered in this work are adaptive in the sense that they can be applied to any classification model. Many ensemble learning approaches assume the same classification model for the base classifiers f_m, $(1 \leq m \leq M)$, but there is no evidence that this performs better than using different ones [42].

At *Feature Level* (Level C), diversity of base classifiers can be achieved by varying the features the individual classifiers are trained with. Breiman [12], for instance, generates a set of uncorrelated decision trees, so-called *random forests*, by randomly selecting a subset of features for training each tree.

The most commonly used approach for ensuring diversity among base classifiers is manipulating the data set (Level D), e.g. by randomly resampling data subsets. This approach has been proven as being very successful and is used by *bagging* and *boosting* algorithms considered in this work.

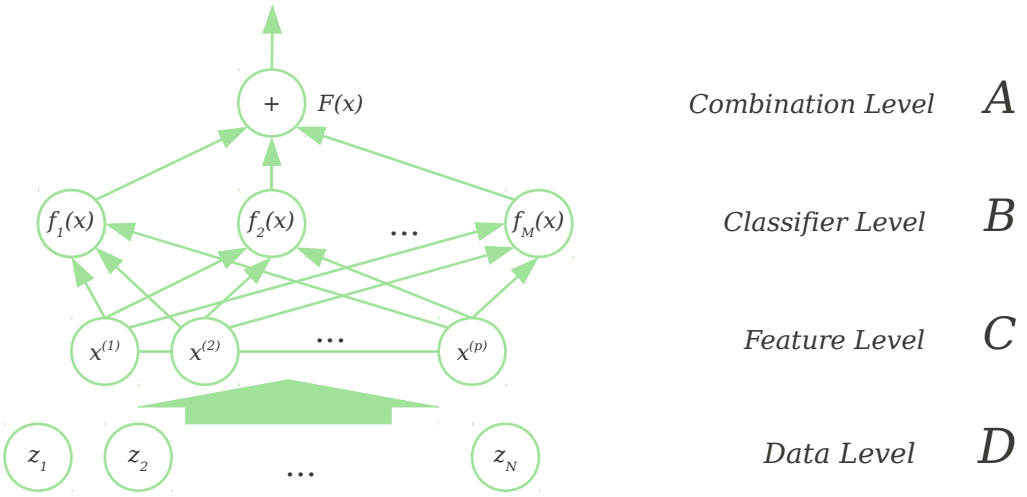


Figure 1.4.: *Schematic representation of an ensemble based classification system on different levels: M "weak" classifiers f_1 to f_M are combined by some combination rule $\oplus$ to get the final decision $F(x)$.*

Combining Classifiers

Having generated a diverse set of classifiers any ensemble system must have a strategy for coming to a final decision by combining the individual votes of the base learners.

There are two main paradigms, namely *fusion* and *selection*, that can be further subdivided along particular aspects. In classifier fusion, each base classifier is supposed to perform well over the whole input/feature space. By contrast, in classifier selection, the members of an ensemble are supposed to be *experts* in a particular subspace of the input space and responsible only for instances that belong to this subspace. One might say, in the latter case, we *select* one or more of the base learners dependently on the data point to classify, whilst in the former case, the final decision is obtained by somehow "averaging" over *all* ensemble members. In practice, there are schemes that lie between these two paradigms, whilst so far, there is no evidence that either performs better in general [26].

Another criterion of a combination rule is *trainability*. Some combiners need no further training after the base classifiers have been learned. A simple majority vote is an example. Others, such as weighted majority voting, need to be trained to determine the weight of each voter.

Some authors [25, 27] make an additional distinction between *data dependent* and *data independent* ensembles. Trainable combiners, in general, are data dependent,

but the dependence can either be implicit or explicit. Implicitly data dependent combiners include methods that train the combiner on the global performance of the data, or, in other words, train the combiner *before* any unseen data point is labeled. Explicit data dependence is observed with combiners that choose the weight of majority voting, for example, dynamically at runtime, dependent on the vicinity of a particular point.

Bagging

Bagging (acronym for ***b**ootstrap **agg**regat**ing***) is a simple and easy to implement ensemble method, introduced by Breiman [18]. In bagging, the ensemble is generated by learning base classifiers from bootstrap [46] replicates of the training set and the individual decisions are combined by majority vote to get the final ensemble output.

Algorithm 1 Bagging

for $m = 1$ to M **do**
  Take a bootstrap sample S_m of the training set $\mathbf{Z} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$
  Fit the classifier $f_m(x) \in \{-1, +1\}$ using S_m
end for
Output the final classifier $\text{sgn}(\sum_{m=1}^{M} f_m(x))$

Algorithm 1 shows the learning and operating procedure of bagging. The diversity of base classifiers f_m necessary to make the ensemble work is achieved by randomly resampling subsets with replacement [46]. Therefore, the base classifiers should have large variance in the sense that they should be sensitive to variations in the data, i.e. small changes in the training set imply large changes in the prediction model. If the base learners are robust against these variations the ensemble consists of many classifiers that are almost identical and the diversity postulate for ensembles is violated, such that an improvement by the committee is quite unlikely. *Classification and Regression Trees* (*CART*) [47] are examples for learning algorithms with high variance.

It is recommended [23] to use relatively large portions of the entire training set for the replicates to ensure that there are sufficient training samples in each subset. Breiman [18] reports on good performance of bagging, which he explains by showing that the mean-squared error of the aggregated predictor is upper-bounded by the mean-squared error averaged over all replicates, and depends on the instability of the weak learner. Therefore, bagging can also be seen as a variance reduction technique on the base classifiers. Domingos [48] gives an explanation in terms of Bayesian

learning theory, by showing that bagging manages to approximate the posterior class probabilities $P(y_k \mid x)$ and shifts the prior distribution of classifier models towards models that have higher complexity.

Since in bagging all base classifiers are generated independently, bagging is a parallel algorithm in the learning as well as in the operating phase, such that the M ensemble members can be distributed on multiple processors.

There are several variants of bagging approaches, the most prominent ones of which are *random forests* and *pasting small votes*.

In [18], Breiman proposes a generalized framework of ensemble models, which he called *random forests*. A random forest consists of an ensemble of decision trees that have been generated by i.i.d. (independent identically distributed) random vectors Θ_k, $(1 \leq k \leq M)$ in a certain strategy. The usage of the vectors Θ_k for ensuring diversity is not further constrained, such that it can be used for sampling from the feature set, the data set, or just randomly varying some parameters of the tree, for example. One of the most successful strategies with random forests has been proven to be random feature selection at each node of the tree [42]. The final decision of the ensemble is obtained by majority voting.

Pasting small votes [49] was designed to be used with large data sets that are divided into smaller subsets, called *bites*, which are used to train the base learners. There are two derivations of pasting small votes, namely *Rvotes* and *Ivotes*, the former of which generates the base learners in parallel, whereas the latter creates consecutive datasets based on their importance, similarly to boosting, which we address in the next section. In the Ivotes variant, the training set for the base classifier in the next iteration $m+1$ is generated by evaluating the current ensemble $F_m(x)$ on any instance x that has not yet been used for training. If the current ensemble fails in predicting its class label, it is definitely placed in the training set for the next classifier. If it succeeds, it is put into the training set only with probability $\varepsilon_m/1-\varepsilon_m$, where ε_m is the error of the m-th classifier. This way, the training set in the next iteration consists of a balanced distribution of "hard" and "easy" training patterns. Pasting small votes is similar to boosting in the manner that the samples that are "harder to classify" are passed to the next iteration.

Boosting

Unlike bagging, which generates the base classifiers in parallel, the *AdaBoost* (acronym for ***Ada**ptive **Boost**ing*) algorithm by Freund and Schapire [6, 7] incrementally fits classifiers, one at a time. The classifier, which is added to the ensemble at iteration

Algorithm 2 General AdaBoost

Start with weights $w_1^{(n)} \leftarrow 1/N,\ n = 1, \ldots, N$
for $m = 1$ to M **do**
Fit the classifier $f_m(x) : \mathcal{X} \mapsto \mathbb{R}$ using weights w_m
Choose $\alpha_m \in \mathbb{R}$
Update the weights:

$$w_{m+1}^{(n)} \leftarrow w_m^{(n)} \cdot e^{-\alpha_m y_n f_m(x_n)} / Z_m \tag{1.10}$$

with normalization factor Z_m, such that $\sum_{n=1}^{N} w_{m+1}^{(n)} = 1$
end for
Output the final classifier $\operatorname{sgn}(F(x))$
with

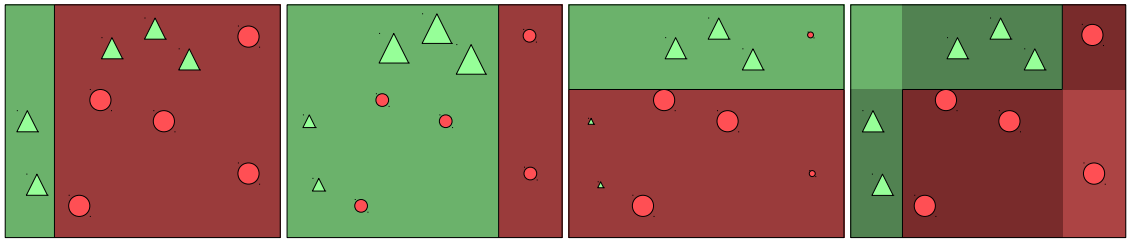
$$F(x) = \sum_{m=1}^{M} \alpha_m f_m(x) \tag{1.11}$$

m is trained on a strategically resampled subset of the original training data. The selection strategy is incorporated in a weight vector $w = (w^{(1)}, \ldots, w^{(N)})$ of length N, which is normalized such that all weights sum to 1, i.e. each pattern in the training data is associated to a weight $w^{(n)}$, which are all initialized uniformly (initially $w_1^{(n)} = 1/N,\ 1 \leq n \leq N$). Resampling in each iteration is done by randomly drawing samples proportionally to the weight distribution.

The key idea of AdaBoost is to increase the weights of samples that have been misclassified by classifier f_m in the current iteration m and to decrease the weights of correctly classified samples, such that the samples that are "hard" to classify are overrepresented in the training data of the next iteration and the subsequent base learner is forced to concentrate more on these patterns.

Additionally, each weak learner is associated to a weight α_m itself, which controls the influence of the m-th classifier on the ensemble decision. The final classifier output is obtained by a weighted majority vote of all base learners.

There are two possible implementations of AdaBoost regarding the generation of the training set in each iteration. The above description refers to weight proportional *resampling*. However, if the base learning algorithm is able to cope with weights in a direct way, the resampling step can be dropped. The implementation of *reweighting* can be seen as a way of "smooth resampling" and makes the algorithm completely



(a) $\epsilon_1 = 0.30, \alpha_1 = 0.42$ ***(b)*** $\epsilon_2 = 0.21, \alpha_2 = 0.65$ ***(c)*** $\epsilon_3 = 0.14, \alpha_3 = 0.92$ ***(d)*** $F(x)$

Figure 1.5.: *2-dimensional toy example; (a)-(c) the decision stumps learned via AdaBoost over 3 iterations. The sizes of the data points correspond to the sizes of their weights (d) the combined ensemble output.*

deterministic. In the listing of Algorithm 2 we assume the latter case and we do so in the rest of this work as well.

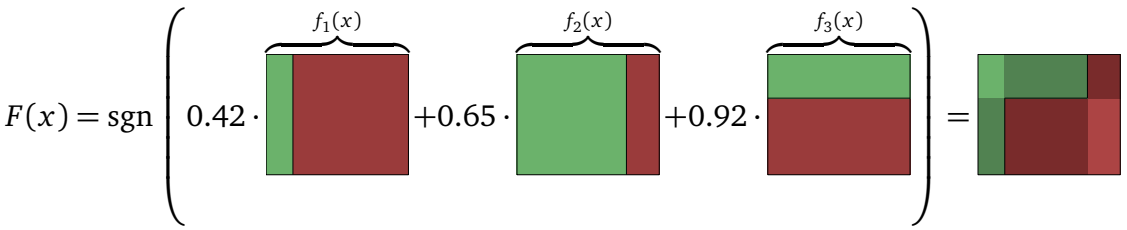
Typically, the α coefficients of each base learner are computed by a logarithmic transform of its weighted error ratio. In particular, it is defined as

$$\alpha_m = \frac{1}{2} \ln \frac{1 - \epsilon_m}{\epsilon_m}, \qquad \text{where } \epsilon_m = \sum_{n=1}^{N} w_m^{(n)} \mathbf{1}_{y_n \neq f_m(x)}, \tag{1.12}$$

such that a base classifier $f_m(x)$ which produces only small weighted error (i.e. $\epsilon_m \to 0$) receives a large weight α_m. In contrast, if it does not better than random guessing (i.e. $\epsilon_m \to 0.5$) its associated weight becomes 0.

In order to illustrate how AdaBoost works in practice, consider the toy example in Figure 1.5. As can be clearly seen, the two-dimensional binary classification problem is not linearly separable. We will show how it still can be solved by AdaBoost, even if only a linear classifier is available. In this example, we use decision stumps as base classifiers, which are linear discriminants, parallel to one axis of the coordinate system. The decision boundary (in this case a simple threshold) of each stump is chosen such that its weighted misclassification error is minimized. Figure 1.5(a) shows the stump that has been learned in the first iteration. It fails in predicting three of the five green samples, so its weighted error is $\epsilon_1 = 0.3$, and, according to (1.12), its corresponding $\alpha_1 = 0.42$ (note that in the first iteration, all weights are equal). Complying the weight update scheme in (1.10), the weights of correctly classified samples are scaled down for the next iteration, whereas the weights of misclassified ones increase. We indicate this by means of the size of data points shown in Figure 1.5, where large-sized data points represent large weight assignments. Since the weight distribution among the samples has been modified, the stump learned in iteration 2 differs from the first one. Figures 1.5(a)−1.5(c) show the decision stumps and the development of sample weights for three iterations as well as corresponding error

rates and α values. According to (1.11) in Algorithm 2, the single hypotheses $f_m(x)$, given by the decision stumps, are combined by a weighted majority vote in order to obtain the final ensemble decision:



The resulting decision boundary is shown in Figure 1.5(d). Although not stuck to a particular family of learning algorithms, the base classifiers often are considered to implement coarse *rules of thumb* [9], i.e. only rough estimates supposed to have only poor individual predictive performance, just slightly better than random guessing. Hence, one often comes across the terms *weak learners* and *weak hypotheses*. In this work these terms are used interchangeably.

Which one is better?

Since with bagging and boosting two ensemble methods have emerged that attract a great deal of attention in the machine learning community, the question arises, which of the two finally performs better. Although there is no "best" method for all problems and we can always construct an instance of a problem on which one of the algorithms performs poorly, the general consensus is that boosting achieves lower generalization error [42]. Indeed, boosting is often regarded as one of "the most powerful learning ideas introduced in the recent 20 years" [40], and Breiman (NIPS Workshop, 1996) even refers to the *AdaBoost* algorithm with trees as the "best off-the-shelf classifier in the world". Since its introduction in 1996 elaborate research has been done in order to understand and refine the performance of AdaBoost, such that boosting has emerged to the probably most growing subarea of classifier composition [42].

One reason, which definitely is accountable for the magnificent success of AdaBoost is its simplicity and ease of implementation. But additionally, some interesting phenomena like its apparent tenacity against overfitting and the ongoing test error reduction, though errorless training performance, undoubtedly contribute to the popularity of boosting. On these grounds, and due to the fact that numerous boosting variants exist, such as cost-sensitive and regularized extensions that put themselves forward to be studied in presence of imbalanced classes, the focus of this work is on the AdaBoost algorithm and its derivatives.

1.2. The Class Imbalance Problem

The pervasive majority of existing classification algorithms are designed for maximizing the overall classification accuracy or minimizing the overall misclassification error in (1.4), respectively. The class imbalance problem is one of the problems that have emerged in recent years, as more and more researchers realized that most of the real-world data sets they have been working with exhibit an imbalance in class distributions leading to suboptimal classification performance [19]. In such cases, almost all patterns are labeled as one class, while far fewer instances are labeled as the other class, which usually is the more important one. Learning from such imbalanced or skewed data sets typically results in a classifier that is biased towards the prevalent class, which directly affects the recognition rate of the minority class and yields misleading statements on classification performance [24].

In this section, we characterize the problems one may encounter in dealing with imbalanced classes and describe techniques for handling these, which are reported in literature.

Nature of the Problem

We can differentiate between two major kinds of imbalance. On the one hand, imbalance may arise as an intrinsic property of the problem. In many applications, such as fraud and intrusion detection or medical diagnosis, the problem of skewed classes is prevalent and due to the rareness of the respective (positive) cases or elaborate elicitation of data and associated high costs, as for crash and fire detection, for instance. In these cases we think of the imbalance problem in terms of uneven a-priori class distributions resulting from the numerical dominance of one of the classes' training instances. In practical applications, the ratio of the small to the large class can be drastic, such as 1:100 or even more, such as the intrusion detection (IDS) data set considered in this work (among others). This subcase of imbalance is often referred to as *class rarity* [20].

However, class imbalances also may occur in domains which are not necessarily intrinsically imbalanced. We can also think of class imbalance as there are uneven costs of making different errors, which can vary per case. In this case, the imbalance results in a specific goal to be accomplished with the classification, reflecting the users' preference for a particular class.

Both kinds of imbalance are not mutually exclusive, but may also occur in parallel, such as in medical diagnosis applications, for example. Here, the benign samples typically overwhelm the malignant cases numerically, such that we expect a classification model to be biased towards the benign class. However, classifying a benign case as malignant is much more desirable than not discovering a disease at all. Hence, in this case, both kinds of imbalance are present. As another example, consider an airbag control system, which has to decide whether or not to trigger the airbag in case of a collision. In this case, undesirably deploying the airbag is off-limits. Equivalently, industrial state-of-the-art fire detection systems have to decide whether to raise the alarm or not, based on the sensory inputs. In such a system, raising a false alarm goes with considerable costs and therefore must be avoided by all means.

There are several terms found in literature class imbalance is referred to. In most of the other work on the topic of imbalance, it is assumed that the positive class ($y = +1$) is always the minority class, or the more important one, respectively. In this work, we pick up on this convention and therefore the terms *minority class*, *positive class* and *small class* are used equivalently.

Difficulties with Imbalance

Weiss [20] gives a survey of the problems arising in dealing with imbalanced classes.

As pointed out earlier in this chapter, evaluation metrics play a crucial role in machine learning. Metrics are used to guide in design of learning algorithms and to evaluate results. Empirical studies [53] have shown that traditional evaluation metrics, such as accuracy, lead to poor minority-class performance because the minority-class examples are much less likely to be predicted than the majority-class examples. Although classification accuracy is close to 100%, precision and recall of the minority class may be far lower, often even close to 0. So, in order to adequately assess a classifier's performance, measures such as F-score or AUC, which are not biased towards the prevalent class, are a more appropriate choice.

Absolute rarity is another fundamental problem associated with imbalanced classes. Here, a lack of data in absolute numbers causes rarity, and rare cases have a much higher misclassification rate than common cases. This problem is often referred to as the problem of *small disjuncts* [20]. Small disjuncts may arise from cases that indeed represent rare or exceptional cases, but also by something else, such as noise, for instance. Most learning algorithms are equipped with some regularization means for preventing the learner from overfitting by removing patterns that are not believed to be "meaningful". In presence of absolute rarity, meaningful patterns are likely to be

identified as noise, which makes separation between noise and meaningful patterns much more difficult than in dealing with large disjuncts.

Relative rarity mainly affects unsupervised learning and describes the problem of rare entities that are hard to find. Weiss [20] illustrates this kind of problem by the phrase "like a needle in a haystack", which is hard to find due to the large number of strands of hay in the haystack. One reason, why relative rarity causes problems, is that rare entities often depend on the conjunction of many conditions, and examining only one condition at a time may not provide sufficient information.

Learning algorithms often employ a *divide-and-conquer* approach, which decomposes the original problem into smaller subproblems that are easier to solve. The CART [47] algorithm is one example, which recursively partitions the input space into smaller cuboid subregions. This process causes *data fragmentation*, which leads to problems since small disjuncts, as they provide little data, may be further fragmented and regularities can only be found within a particular subregion. So divide-and-conquer algorithms may particularly suffer from rare cases.

Solutions

A number of solutions to the class imbalance problem have been previously proposed both at the data level as well as the algorithmic level. At the data level, the objective is to rebalance the class distributions by resampling the training data. At the algorithmic level, the goal is to make existing learning algorithms sensitive with respect to the minority class.

Data-level Approaches

The main idea in countering the imbalance problem from data level is to make the data set balanced by a certain resampling technique. Approaches at the data level include many different forms of resampling, such as randomly oversampling the minority class or randomly undersampling the majority class. The resampling can be purely random or informative. In informative resampling, the selection of particular samples is targeted. For example, in informatively undersampling the majority class, instances may be dropped that are easily separable from the minority class [24].

Although resampling is relatively easy to implement and therefore a widely used method in dealing with imbalanced classes, there remain some difficulties that may lead to suboptimal results. Weiss *et al.* [54] empirically showed that, regarding

the classification performance on the minority class, a prior distribution of 50:50 is preferable and often yields good results, but is not necessarily optimal.

There are other difficulties with resampling. If we choose to undersample the prevalent class, for example, we run onto risk that we omit meaningful patterns that could be important in the learning phase. If we oversample the rare class, it has been reported by several authors [55, 56] that the likelihood of overtraining the classifier increases, since exact replicates of the rare cases are made. Additionally, by oversampling, the computational expense may increase if we are dealing with already large, but imbalanced data sets [24].

Algorithmic Approaches

A common strategy for approaching the imbalance problem at the algorithmic level is based on choosing an appropriate inductive bias towards the minority class [21].

For confidence-rated classifiers, i.e. models returning a real-valued score representing the degree to which a pattern is believed to be a member of a particular class, such a bias can be induced after the learning process has finished by varying the decision threshold separating the classes [20].

Another way is to introduce class-specific misclassification costs and to make a learning algorithm cost-sensitive. The goal is to minimize misclassification costs during the learning phase, based on a predefined cost matrix. The main shortcoming of cost-sensitive approaches is, as we will discuss more detailed in Chapter 2, that costs are not always available.

One-class learning can be very useful being applied to extremely imbalanced data sets composed of a high-dimensional, noisy feature spaces [57]. A one-class learning task is characterized by the fact that there are only training data available belonging to one class. A one-class classifier is then to predict whether or not a given example belongs to that class. An example from information retrieval, for instance, is the task of classifying web sites that are of interest to a web user only based on the history in his browser.

1.3. Related Work

A lot of work has been published in the recent decade on the topic of learning from imbalanced data sets. Indeed, becoming aware of the importance of this topic caused a rapid increase in interest, which gave rise to two workshops in 2000 [1] and 2003 [2] at the AAAI[1] and ICML[2] conferences, as well as a Special Issue on Class Imbalances in the ACM SIGKDD[3] Explorations Newsletter in 2004 [4]. A wide survey of strategies for handling imbalance is given by Kotsiantis *et al.* [24] and He and Garcia [5].

As mentioned above, a widely used approach for countering class imbalances is inducing a bias towards the minority class into the prediction model. Japkowicz [13] discusses two strategies for rebalancing class distributions, i.e. over-sampling the minority class and under-sampling the majority class. Maloof [16] shows that under-sampling one class before learning takes place is equivalent to varying the ROC decision threshold afterwards. Both authors agree that under-sampling the majority class works better than over-sampling the minority class. These results are confirmed by Drummond and Holte [17], who empirically study sampling techniques in presence of class imbalance and uneven misclassification costs in context of the *C4.5* algorithm by Quinlan [78].

In context of boosting, resampling techniques have been proposed by several authors. Chawla *et al.* [35] present their *SMOTEBoost* (***S**ynthetic **M**inority **O**versampling **TE**chnique*) algorithm which generates new synthetic samples along the line segment connecting a sample of the minority class and its nearest neighbor. A different resampling technique is proposed by Guo and Viktor [29], which creates synthetic samples from patterns that are difficult to classify, which they call *seed examples*. Both approaches differ in the strategy for generating new samples, the underlying AdaBoost procedure, however, is unaffected.

In this thesis, instead of resampling procedures, we limit our scope to algorithmic approaches for countering imbalance in boosting. As a common technique, several authors discuss introducing cost items for uneven misclassification costs. Sun *et al.* [21] introduce their AdaC1, AdaC2 and AdaC3 algorithms by inserting cost factors at different positions in the boosting weight update scheme, and several other cost-sensitive extensions of boosting have been proposed by Ting *et al.* [36], Li *et al.* [28] and Fan *et al.* [30]. In Chapter 2 we study each of these algorithms in detail.

[1] Association for the Advancement of Artificial Intelligence
[2] International Conference on Machine Learning
[3] Association for Computing Machinery, Special Interest Group on Knowledge Discovery and Data Mining

RareBoost by Joshi *et al.* [32] is a variant of the AdaBoost algorithm that differentiates between class-specific misclassifications, which, in presence of skewed classes, has been proven to significantly outperform original AdaBoost. A slightly modified version of RareBoost has been proposed by Song *et al.* [33].

Rätsch [37] points out several similarities of boosting and Support-Vector-Machines [51]. He adapts the concept of soft-margins for regularization used in SVMs to AdaBoost, giving raise to the AdaBoost$_{Reg}$ algorithm. Another regularized boosting method has been proposed by Liu *et al.* [38].

For a thorough analysis of the algorithms just mentioned we refer to Chapter 2 of this thesis.

1.4. Thesis Contributions

As pointed out in the previous section, there has been a lot of research in the field of boosting in general and particularly in the field of cost-sensitive boosting variants for coping with imbalanced classes. A further line of research are regularized boosting approaches for dealing with noise and overlapping class distributions. However, the combination of both, i.e. the effects of regularized boosting in case of class imbalance and the impact of employing cost-sensitive boosting techniques in noisy settings, is rarely addressed in literature.

In particular, in our work we address the following issues:

1. In Chapter 2 we give a detailed survey of current boosting techniques.
2. We theoretically and empirically analyze AdaBoost's sensitivity to noise and validate recent studies that refute the presumption that AdaBoost seems to be immune against overfitting.
3. We show that the only boosting algorithm tackling the imbalance problem without using explicit costs, namely RareBoost, does not solve the problem, but is indeed a special case of the well known RealBoost algorithm, a probabilistic variant of AdaBoost.
4. We show that particularly regularized learning algorithms suffer from the imbalance problem and present some novel insights on the topic of cost-sensitive variants, which tend to overfit.

The remainder of this thesis is organized as follows: In Chapter 2 we give a detailed survey of the current state-of-the-art in boosting, studying some of the most

outstanding properties of the AdaBoost algorithm and reviewing some of its modifications, such as extensions to cost-sensitivity and regularization.

In Chapter 3, we empirically substantiate our findings by presenting the results obtained by experiments that have been conducted with several real-world data sets from an industrial environment as well as academic benchmarks. We assess the algorithms' performance in practice and identify methods that we cannot recommend.

Chapter 4 briefly presents the Matlab toolbox, which has been implemented in context of this thesis and comprises 14 of the boosting approaches considered in this work.

Chapter 5 concludes this work and gives an outlook of remaining issues and future investigations.

Chapter 2

A Survey of Boosting

In this chapter we give an overview of the current state-of-the-art in boosting. We start with discussing some of the most important properties of the AdaBoost algorithm, and then review several boosting modifications that have been proposed in literature. We additionally present novel insights concerning some of the algorithms, in particular regarding cost-sensitive variants and approaches countering the imbalance problem.

2.1. General Properties

In the AdaBoost algorithm, the final classifier $F(x)$ is given by a weighted majority vote of all single weak hypotheses,

$$F(x) = \sum_{m=1}^{M} \alpha_m f_m(x), \tag{2.1}$$

where f_m is the weak hypothesis learned in the m-th iteration and α_m is its associated weight. We denote the number of iterations or the number of base classifiers in the ensemble as M. In this section, we concentrate on the original AdaBoost algorithm as introduced in Chapter 1 and derive some of its interesting theoretical properties, which are able to explain why AdaBoost works.

Training Error

One of the most basic theoretical properties of AdaBoost is its ability to scale down the training error, i.e. the rate of misclassified samples in the training set. In [31],

Schapire and Singer show that the training error rate of the final classifier, given by (1.4), is bounded above by an exponential error rate. What easily can be seen in Figure 2.2 on page 34 by comparing the exponential and the misclassification loss functions, can be shown theoretically by first noting that, if a given sample x_n in the training set is misclassified, i.e. $F(x_n) \neq y_n$, where $y_n \in \{-1, +1\}$, then the product of classifier output and class label is non-positive:

$$y_n F(x_n) = y_n \sum_{m=1}^{M} \alpha_m f_m(x_n) \leq 0,$$

implying that

$$e^{-y_n F(x_n)} \geq 1, \quad \text{hence} \quad \mathbf{1}_{F(x_n) \neq y_n} \leq e^{-y_n F(x_n)}. \tag{2.2}$$

So we can assess the error rate over the whole training set as

$$\text{training\_error}(F) = \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}_{F(x_n) \neq y_n} \quad \leq \quad \frac{1}{N} \sum_{n=1}^{N} e^{-y_n F(x_n)}. \tag{2.3}$$

Furthermore, we can show how this upper bound on the training error is related to the sample weights. From (1.10) in Algorithm 2 we know that the sample weights in each iteration are updated by $w_{m+1}^{(n)} = \frac{1}{Z_m} w_m^{(n)} e^{-y_n \alpha_m f_m(x_n)}$, where Z_m is a normalization factor, such that the weights sum up to one. By unraveling the update rule over all of the M iterations, the final weights w_{M+1} at the end of the training process are given by

$$w_{M+1}^{(n)} = \frac{1}{\prod_{m=1}^{M} Z_m} e^{-\sum_{m=1}^{M} \alpha_m y_n f_m(x_n)} = \frac{1}{\prod_{m=1}^{M} Z_m} e^{-y_n F(x_n)}, \tag{2.4}$$

and plugging (2.4) into (2.3) yields

$$\begin{aligned} \text{training\_error}(F) \quad &\leq \quad \frac{1}{N} \sum_{n=1}^{N} e^{-y_n F(x_n)} \\ &= \quad \sum_{n=1}^{N} \left(\prod_{m=1}^{M} Z_m \right) w_{M+1}^{(n)} \\ &= \quad \prod_{m=1}^{M} Z_m. \end{aligned} \tag{2.5}$$

Eq. (2.5) shows that the overall misclassification error of (1.4) in an arbitrary iteration M is upper bounded by the product of all normalization factors Z_m of previous iterations ($1 \leq m \leq M$). This suggests that the training error of the ensemble classifier can be reduced in each iteration by minimizing the normalization factors

most rapidly by choosing the weak hypothesis $f_m(x)$ and its corresponding weight α_m in a neat way. Suppose we have a weak learner providing binary output, i.e. $f_m(x) \in \{-1, +1\}$. Further splitting Z_m (which is given by the sum of all weights in the current iteration) into correct and incorrect predictions yields

$$Z_m = \sum_{n=1}^{N} w_m^{(n)} e^{-y_n \alpha_m f_m(x_n)} = \sum_{m=1}^{M} w_m^{(n)} \left(\frac{1 + y_n f_m(x_n)}{2} e^{-\alpha_m} + \frac{1 - y_n f_m(x_n)}{2} e^{\alpha_m} \right) \quad (2.6)$$

The transformation is valid since for each case $y_n f_m(x_n) = \pm 1$, either of the terms in brackets vanishes. In a next step we analytically minimize the righthand side of (2.6) with respect to α_m and get

$$\alpha_m = \frac{1}{2} \ln \frac{1 - \epsilon_m}{\epsilon_m}, \qquad \text{with } \epsilon_m = \sum_{n=1}^{N} w_m^{(n)} \mathbf{1}_{y_n \neq f_m(x_n)}. \quad (2.7)$$

as the optimal choice for α_m regarding the base learner. For a proof we refer to Lemma 4 in Appendix A. The resulting algorithm, which afterwards was given the name "Discrete AdaBoost", is listed in Algorithm 3. There are variants using real-valued base classifiers instead of binary ones, which have been proposed two years after AdaBoost's inception [31, 11]. We will introduce them later in this chapter.

So far, we have only discussed how to choose α_m in an appropriate way. However, there is another model parameter that is to be determined in each iteration for reducing the training error, namely the weak learner. In order to understand the conditions to be met by a weak hypothesis to be efficiently boosted, we note that the following bound holds on Z_m:

$$\frac{1}{N} \sum_{n=1}^{N} \mathbf{1}_{F(x_n) \neq y_n} \leq \prod_{m=1}^{M} Z_m \leq \exp\left(-2 \sum_{m=1}^{M} (1/2 - \epsilon_m)^2 \right), \quad (2.8)$$

where ϵ_m is the misclassification error produced by the base learner. A proof for this bound was first provided by Freund and Schapire [7]. The central property we can derive from (2.8) is that, as long as the base learner $f_m(x)$ we use can do just slightly better than random guessing, i.e. $\epsilon \neq 0.5$, then the training error in (2.8) will drop down exponentially fast over the M iterations. It is reasonable to demand the weighted error rate to be $\epsilon \neq 0.5$ instead of $\epsilon < 0.5$ since a classifier with error > 0.5 can be converted into one with error < 0.5 by inverting its output. The authors of [7] regard this as the proof that AdaBoost is indeed a "true" boosting algorithm in the sense that it is able to efficiently convert a weak learning algorithm into a strong one with an arbitrarily low error rate.

Algorithm 3 Discrete AdaBoost

Start with weights $w_1^{(n)} \leftarrow 1/N,\ n = 1, \ldots, N$
for $m = 1$ to M **do**
Fit the classifier $f_m(x) \in \{-1, +1\}$ using weights w_m
Compute

$$\epsilon_m = \sum_{n=1}^{N} \mathbf{1}_{y_n \neq f_m(x_n)} w_m^{(n)} \tag{2.9}$$

$$\alpha_m = \frac{1}{2} \ln \frac{1 - \epsilon_m}{\epsilon_m} \tag{2.10}$$

Update weights

$$w_{m+1}^{(n)} \leftarrow \frac{w_m^{(n)} e^{-y_n \alpha_m f_m(x_n)}}{Z_m} \tag{2.11}$$

where Z_m is a normalization factor, such that $\sum_{n=1}^{N} w_{m+1}^{(n)} = 1$
end for
Output the final classifier $\mathrm{sgn}(F(x))$, where

$$F(x) = \sum_{m=1}^{M} \alpha_m f_m(x) \tag{2.12}$$

Generalization Error

Of course, optimizing the progress of the training error a classifier produces during its learning phase is only a secondary goal in classifier design. As pointed out in Chapter 1, the main concern in machine learning is the generalization performance of models, i.e. how they behave when being faced with unknown data points.

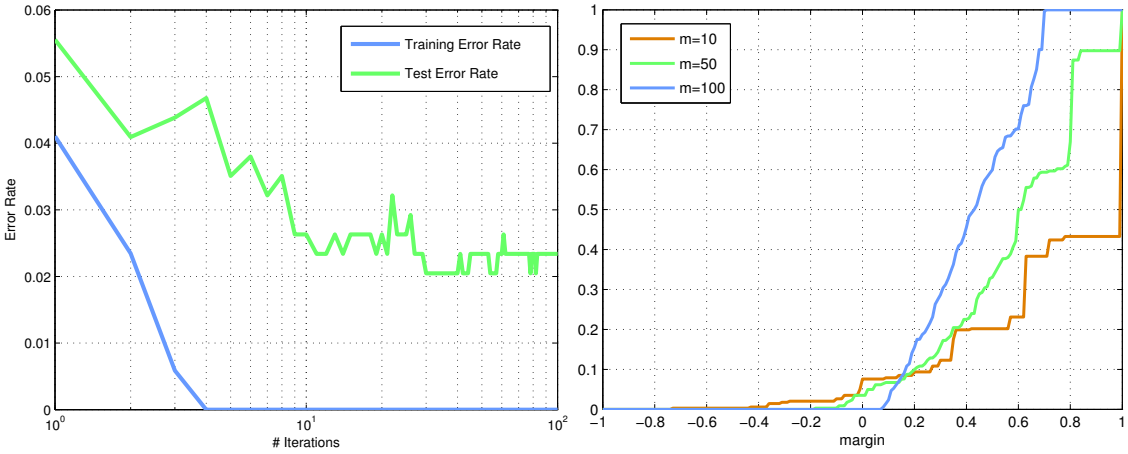
In literature, two main approaches for analyzing the generalization performance of AdaBoost have been proposed. The first, provided by Freund and Schapire [7] in 1997, is based on theory of the *Vapnik-Chervonenkis (VC) dimension* [59, 60]. The authors show that, assuming that test and training set consist of randomly, i.i.d. drawn samples from some unknown joint distribution of features X and labels y, the generalization error, i.e. the probability of misclassifying a new sample, is at most

$$\hat{P}(F(x)y < 0) + \tilde{O}\left(\sqrt{\frac{Md}{N}}\right), \tag{2.13}$$

where $\hat{P}(\cdot)$ denotes the empirical probability on the training set, d the VC dimension of the base classifier and $\tilde{O}(\cdot)$ the "Soft-O"-notation (which is identical to the classical O-notation, except that logarithmic *and* constant factors are hidden).

However, in practice, the bound in (2.13) has been empirically proven as being too loose to be of practical value by several authors [61, 65, 66]. Eq. (2.13) suggests that, once the training error probability of the ensemble has reached its minimum, i.e. $\hat{P}(F(x)y < 0) = 0$, the generalization error increases with each additional iteration, indicating a tendency to overfitting. Indeed, in empirical evaluations, it is often observed that AdaBoost keeps scaling down the generalization error even after the training error has reached zero. This is a remarkable property since it contradicts the *Occam's Razor* principle, which suggests that, whenever we can choose between multiple hypotheses which are equal in all other aspects, we do best when selecting the simplest one. An example of such behavior is given in Figure 2.1, where we run AdaBoost on the Wisconsin breast cancer benchmark data set over 100 rounds, using decision stumps as weak learners. As can be seen in Figure 2.1(a), the training error reaches zero after 4 iterations, but from then, the test error keeps decreasing half as much.

Hence, since the above upper bound appears quite inaccurate, we omit discussing it in further detail and move on to the second way of analyzing AdaBoost's generalization performance that has been proposed in literature.



(a) *Training error vs. test error* ***(b)*** *Cumulative margin distribution*

Figure 2.1.: *AdaBoost on Wisconsin Breast Cancer Benchmark: 2-fold cross validation over 100 runs with decision stumps. (a) training and test error (b) cumulative margin distribution at* $m \in \{10, 50, 100\}$ *iterations.*

Boosting the Margin

In 1998, Schapire *et al.* [10] came up with an alternative explanation for the effectiveness of the AdaBoost algorithm, employing the concept of *margins* known from support vector classifiers. In the boosting context, the margin of a sample (x, y) is defined as

$$\text{margin}(x, y) = \frac{yF(x)}{\sum_{m=1}^{M} |\alpha_m|} = \frac{y \sum_{m=1}^{M} \alpha_m f_m(x)}{\sum_{m=1}^{M} |\alpha_m|} \in [-1, +1] \tag{2.14}$$

and can be interpreted as a measure of the "confidence" of the prediction. For correctly classified instances the margin is positive and negative for incorrectly predicted ones, respectively. Schapire *et al.* show that larger margins guarantee lower generalization error and give a new upper bound

$$\hat{P}(\text{margin}(x, y) \leq \theta) + \tilde{O}\left(\sqrt{\frac{d}{N\theta^2}}\right), \tag{2.15}$$

for the probability that the ensemble will misclassify a new example for $\theta > 0$, with all variables as defined above. A notable property of (2.15) is that it is entirely independent of M, the number of iterations, which, given the empirical data in Figure 2.1(a), seems to be a more reasonable bound than (2.13). Additionally, as can be seen in the weight update scheme of AdaBoost (Eq. (2.11)), the algorithm pursues a particularly aggressive strategy for increasing the margin, since in each round of

boosting, it concentrates on the training patterns with smallest (i.e. negative) margin and increases their weights exponentially fast. Figure 2.1(b) shows the progress of the cumulative margin distribution of training patterns at different iterations. It can be seen that, with increasing number of iterations, the margins increase as well.

Robustness

Due to its aggressiveness towards misclassified samples the question arises how AdaBoost behaves in presence of outliers, noise or overlapping class distributions. In this section we discuss the robustness of AdaBoost against such extreme data and its influence on the training process.

In the previous section we derived the original AdaBoost algorithm by minimizing a differentiable upper bound on the training misclassification error, as AdaBoost was originally motivated by Freund and Schapire [7]. However, as already pointed out, in practice, these bounds turn out to be too loose to have practical value. In order to better understand its performance, Friedman *et al.* [40] give a more intuitive view on boosting, showing that AdaBoost fits an additive model by stagewise minimization of an exponential loss function.

Additive Models

An *additive model* takes the form

$$F(x) = \sum_{m=1}^{M} \alpha_m f(x; \gamma_m), \tag{2.16}$$

where $f(x;\gamma)$ are a set of elementary basis functions that take a multivariate argument x, and are characterized by some parameters γ. The α_m, $m = 1,\ldots,M$ are mixing coefficients determining a weight for each basis function. Typically, a model $F(x)$ is learned from a data set by minimizing some loss function L, such that

$$\min_{\{\alpha_m,\gamma_m\}_1^M} \sum_{n=1}^{N} L\left(y_n, \sum_{m=1}^{M} \alpha_m f(x_n;\gamma_m)\right). \tag{2.17}$$

For many loss and basis functions, optimizing (2.17) globally, i.e. finding the joint of all (α_m,γ_m) pairs that minimizes the loss function at once is computationally intractable. However, often a reasonable approximation can be found by sequentially optimizing only one single basis function without adjusting any of the model parameters and coefficients that have already been learned. This leads to the "greedy"

Algorithm 4 Forward Stagewise Additive Modeling

Initialize $F_0(x) = 0$
for $m = 1$ to M **do**
Compute

$$(\alpha_m, \gamma_m) = \arg\min_{\alpha,\gamma} \sum_{n=1}^{N} L(y_n, F_{m-1}(x_n) + \alpha f(x_n; \gamma)). \tag{2.18}$$

Set $F_m(x) = F_{m-1}(x) + \alpha_m f(x; \gamma_m)$.
end for

forward stagewise additive modeling procedure shown in Algorithm 4. In each iteration m, there is only one basis function $f(x; \gamma_m)$ to be learned for optimizing the loss function regarding all previously learned terms, which are not modified.

It can be shown that AdaBoost fits an additive model of the form given in (2.16) by forward stagewise additive modeling, where the loss function L is defined as

$$L(y, F(x)) = e^{-yF(x)}, \tag{2.19}$$

i.e. an exponential loss. Since we are dealing with binary problems, i.e. $y \in \{-1, +1\}$, the exponent of the loss function in (2.19) turns positive if the model fails in classifying an example correctly, and becomes negative otherwise. Therefore, and due to the monotonicity of the reciprocal exponential function, incorrectly predicted samples are assigned a much larger loss than correctly predicted ones. Hence, by minimizing (2.17) with L defined in (2.19), we are urging the model to predict the class memberships as correctly as possible.

Plugging (2.19) into (2.18) of Algorithm 4, in each iteration one must solve for

$$(\alpha_m, f_m) = \arg\min_{\alpha,f} \sum_{n=1}^{N} e^{-y_n(F_{m-1}(x) + \alpha f(x_n))}, \tag{2.20}$$

where we have defined $f_m(x) = f(x; \gamma_m)$, and $F_{m-1}(x) = \sum_{i=1}^{m-1} \alpha_i f_i(x)$ denotes the model up to the current iteration. Since for all data points x_n, $F_{m-1}(x_n)$ is assumed to be fixed and neither depends on α nor on f in the m-th iteration, it can be subsumed in a weight $w_m^{(n)}$, such that (2.20) can be rewritten as

$$\begin{aligned} (\alpha_m, f_m) &= \arg\min_{\alpha,f} \sum_{n=1}^{N} e^{-y_n F_{m-1}(x_n)} e^{-y_n \alpha f(x_n)} \\ &= \arg\min_{\alpha,f} \sum_{n=1}^{N} w_m^{(n)} e^{-y_n \alpha f(x_n)}, \end{aligned} \tag{2.21}$$

having defined $w_m^{(n)} = e^{-y_n F_{m-1}(x_n)}$. Splitting the criterion in (2.21) with respect to correctly and incorrectly predicted patterns yields

$$\begin{aligned}(\alpha_m, f_m) &= \arg\min_{\alpha,f} \left(e^{\alpha} \sum_{n=1}^{N} \mathbf{1}_{y_n \neq f(x_n)} w_m^{(n)} + e^{-\alpha} \sum_{n=1}^{N} \mathbf{1}_{y_n = f(x_n)} w_m^{(n)} \right) \\ &= \arg\min_{\alpha,f} \left((e^{\alpha} - e^{-\alpha}) \sum_{n=1}^{N} \mathbf{1}_{y_n \neq f(x_n)} w_m^{(n)} + e^{-\alpha} \sum_{n=1}^{N} w_m^{(n)} \right). \quad (2.22)\end{aligned}$$

It is traceable that, for any given α, (2.22) is minimized, when

$$f_m(x) = \arg\min_{f(x)} \sum_{n=1}^{N} \mathbf{1}_{y_n \neq f(x_n)} w_m^{(n)} \quad (2.23)$$

is minimized with respect to f, since all other terms are constant, which corresponds to the classifier that minimizes the misclassification error taking into account the sample weights of the current iteration $w_m^{(n)}$. Having found the optimal $f_m(x)$, there remains the issue of choosing the corresponding mixing coefficient α_m. Setting the derivative of (2.22) with respect to α to zero yields

$$\begin{aligned} e^{\alpha} \sum_{n=1}^{N} \mathbf{1}_{y_n \neq f(x_n)} w_m^{(n)} &= e^{-\alpha} \sum_{n=1}^{N} \mathbf{1}_{y_n = f(x_n)} w_m^{(n)} \\ \alpha + \ln \sum_{n=1}^{N} \mathbf{1}_{y_n \neq f(x_n)} w_m^{(n)} &= -\alpha + \ln \sum_{n=1}^{N} \mathbf{1}_{y_n = f(x_n)} w_m^{(n)} \\ \alpha_m &= \frac{1}{2} \ln \frac{1 - \epsilon_m}{\epsilon_m}, \quad (2.24)\end{aligned}$$

with ϵ_m being the weighted misclassification error produced by f_m. Corresponding to Algorithm 4, the approximation and weights are then updated by

$$F_m(x) = F_{m-1}(x) + \alpha_m f_m(x), \qquad w_{m+1}^{(n)} = w_m^{(n)} e^{-y_n \alpha_m f_m(x_n)},$$

which is exactly the AdaBoost update scheme in Algorithm 3.

Loss Functions

Up to now, the choice of an exponential loss function seems somehow arbitrary. We have seen that (2.19) is a monotonically decreasing function of the "margin" $yF(x)$, which is positive for correctly classified instances ($yF(x) > 0$), and negative for incorrectly classified ones ($yF(x) < 0$), such that the training process will concentrate on samples with large negative margins. In fact, the principal attraction of using exponential criteria for boosting is computational, because it leads to the simple reweighting update scheme in (2.11) [11].

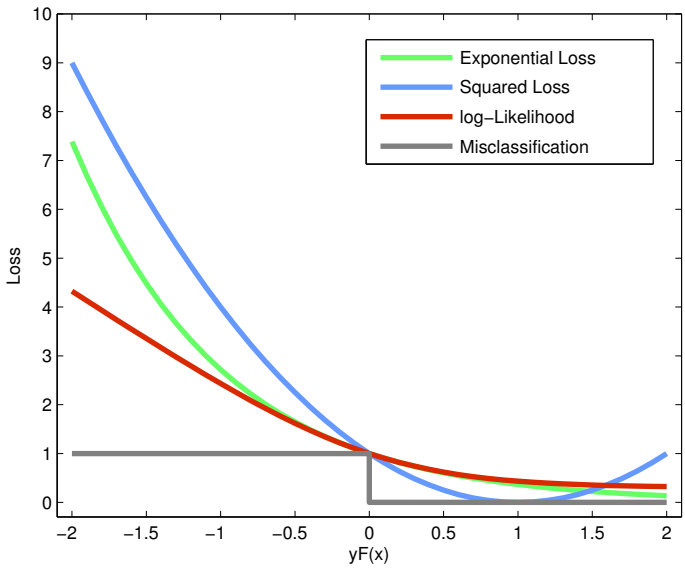


Figure 2.2.: *Loss functions for binary classification, with response* $y = \pm 1$ *and classifier output* $F(x)$*: Misclassification* $\mathbf{1}_{sgn(F(x))\neq y}$*; squared loss:* $(F(x)-y)^2$*; exponential loss:* $e^{-yF(x)}$*; binomial log likelihood:* $\ln(1+e^{-2yF(x)})$*. The functions have been scaled, such that they pass through the point* $(0,1)$

Consider, for example, the situation where we are given data from a noisy environment, where we have overlapping class distributions, such that the Bayesian error is not close to zero, or mislabeled training instances. These phenomena make the classification task difficult and harm a classifier's performance [71]. Typically, in the AdaBoost context, such instances have a negative margin since a base learner, especially one exhibiting large bias, will not pay much attention to such scattered occurrences. However, the exponential loss continuously penalizes large negative margins by increasing their weights exponentially fast, much more than rewarding positive ones by decreasing their weights. As a consequence, at each iteration, noisy data will exponentially gain influence on the training process. It has been empirically observed, as we will confirm in Chapter 3, that the AdaBoost algorithm is indeed sensitive to noise and suffers dramatically from such situations.

So, in order to relax the problem of AdaBoost's sensitivity to noisy data the question of choosing an alternative error loss arises. In Figure 2.2 we plot a selection of possible loss functions of the margin, including the misclassification or 0/1 loss,

$$L(y,F(x)) = \mathbf{1}_{yF(x)<0}, \tag{2.25}$$

the actual misclassification error.

One loss function, which is commonly employed for regression problems, is the *squared error loss*,

$$L(y, F(x)) = (y - F(x))^2. \tag{2.26}$$

The goal of classifier learning is to produce positive margins, i.e. $yF(x) > 0$, as frequently as possible, or, in other terms, to minimize the number of misclassified patterns. Therefore, any loss criterion should penalize errors, i.e. negative margins, much harder than correct predictions, i.e. positive margins, which postulates monotone degression of the loss function with respect to an increasing margin. However, as can be seen in Figure 2.2, the squared error loss is not a monotone decreasing function of the margin, but it rather puts more emphasis on correctly classified patterns (starting from margin $yF(x) > 1$), reducing the (relative) influence on misclassified ones. Hence, we can conclude that squared error is an inappropriate substitute for the exponential loss [11, 40].

The fourth loss function depicted in Figure 2.2 is given by the *cross-entropy* function,

$$L(y, F(x)) = -\ln(1 + e^{-2yF(x)}), \tag{2.27}$$

which is also known as *deviance* or *binomial log-likelihood*. Like the exponential loss, cross-entropy is monotone decreasing in increasing margins, hence penalizes misclassified observations and rewards correct predictions. But, in contrast, penalty for large negative margins grows linearly in $yF(x)$, instead of exponentially, and therefore cross-entropy tends to distribute importance more uniformly over all samples. So, boosting by means of minimizing cross-entropy loss is expected to be far more robust to noisy settings than the classical AdaBoost algorithm. In the next section, we will introduce *LogitBoost*, a boosting algorithm striving to accomplish that.

2.2. Boosting Variants

So far, we presented two common interpretations of boosting which are frequently referred to in literature. In doing so, we approached the boosting task from *data level*, i.e. we concentrated on the available training data and gauged the prediction model by assigning an error loss to each pattern $z_n \in Z$ and by minimizing the cumulative error loss over the whole data set. Friedman *et al.* [11, 40] give a third interpretation of boosting from a more statistical point of view, tackling the problem at *population level*, i.e. instead of explicitly taking into account each particular pattern in a data set, only *one* arbitrary pattern is considered, treated as a random variable.

The statistical view of boosting presented in this chapter provides quite an elegant way of deriving the AdaBoost algorithm on the one hand and, on the other hand, it reveals some very interesting properties of boosting as, for example, its similarity to the logistic regression model. Additionally, it gives rise to a number of modified versions and improvements of the original AdaBoost algorithm.

Logistic Regression

In this section we briefly revisit the logistic regression model for classification. From a probabilistic point of view, the binary classification task can be reformulated to

$$y = \arg\max_{y_k} P(y_k \mid x), \qquad k \in \{1, 2\}, \tag{2.28}$$

which returns the class with highest probability given an observation x, such that, for performing classification, the conditional class probability distribution $p(y \mid x)$ has to be modeled. The *logistic regression* model takes the form

$$P(y = +1 \mid x) = \frac{e^{F(x)}}{e^{F(x)} + e^{-F(x)}}, \tag{2.29}$$

where $F(x)$ is a function of the input vector $x = (x^{(1)}, \ldots, x^{(p)})$. In classical logistic regression, the features $x^{(i)}$ are supposed to be binary, and $F(x)$ implements a linear combination of all features. The final model (i.e. the weights of the linear combination) is usually obtained by employing a maximum likelihood estimate.

Of course, the probability estimates of (2.29) lie in $[0, 1]$, but inverting (2.29), we get the *log odd* or *logit* transform

$$F(x) = \ln \frac{P(y = +1 \mid x)}{P(y = -1 \mid x)}, \tag{2.30}$$

which lies in $\mathbb{R}$. It can be shown [11] that the logistic regression model of (2.29) is a population minimizer of the expected exponential loss conditioned on x,

$$\mathbb{E}(e^{-yF(x)} \mid x). \tag{2.31}$$

Unraveling the expectation we get

$$\mathbb{E}(e^{-yF(x)} \mid x) = e^{-F(x)} P(y = +1 \mid x) + e^{F(x)} P(y = -1 \mid x),$$

and setting the derivative to zero yields

$$\begin{aligned} \frac{\partial \mathbb{E}(e^{-yF(x)} \mid x)}{\partial F(x)} &= -e^{-F(x)} P(y = +1 \mid x) + e^{F(x)} P(y = -1 \mid x) = 0 \\ e^{F(x)} P(y = -1 \mid x) &= e^{-F(x)} P(y = +1 \mid x) \\ F(x) &= \frac{1}{2} \ln \frac{P(y = +1 \mid x)}{P(y = -1 \mid x)}. \end{aligned} \tag{2.32}$$

So, as can be seen, the logistic regression model (2.29) and the minimizer of the conditional exponential loss (2.32) (at population level) are equivalent up to a factor 2. From this equivalence we can conclude that the AdaBoost algorithm, which approximately minimizes the exponential loss, also approximately fits an additive logistic regression model [11].

Discrete AdaBoost

In order to substantiate the argument that AdaBoost additively fits a logistic regression model we can derive the Discrete AdaBoost algorithm by minimizing the expected exponential loss via Newton-like update steps. For this, suppose we are currently in iteration m, having trained a model $F_{m-1}(x) = \sum_{i=1}^{m-1} \alpha_i f_i(x)$ and we are seeking an improved estimate $F_{m-1}(x) + \alpha_m f_m(x)$.

The objective function then becomes

$$J(F_{m-1}(x) + \alpha_m f_m(x)) \quad = \quad \mathbb{E}\left(e^{-y(F_{m-1}(x)+\alpha_m f_m(x))} \,|\, x\right)$$

which is minimized at

$$f_m(x) = \begin{cases} +1 & \text{if } P_w(y=+1\,|\,x) \geq P_w(y=-1\,|\,x) \\ -1 & \text{if } P_w(y=+1\,|\,x) < P_w(y=-1\,|\,x) \end{cases}, \tag{2.33}$$

where $P_w(\cdot\,|\,x)$ refers to a weighted probability estimate with weights defined as $w(x) = e^{-yF_{m-1}(x)}$. A proof for (2.33) can be found in Lemma 5 in Appendix A. For obtaining α_m, the objective function $J(F_{m-1} + \alpha_m f_m(x))$ can be minimized by setting the derivative of $\mathbb{E}\left(e^{-y(F_{m-1}(x)+\alpha_m f_m(x))} \,|\, x\right)$ with respect to α_m to zero, which yields

$$\begin{aligned} \alpha_m &= \arg\min_{\alpha} \mathbb{E}\left(e^{-yF_{m-1}(x)+\alpha f_m(x))}\Big|\, x\right) \\ &= \frac{1}{2}\ln\frac{1-\epsilon_m}{\epsilon_m}, \qquad \epsilon_m = \mathbb{E}_w(\mathbf{1}_{y\neq f_m(x)}\,|\,x), \end{aligned} \tag{2.34}$$

where the notation $\mathbb{E}_w(\cdot\,|\,x)$ refers to the *weighted conditional expectation*, which is defined as

$$\mathbb{E}_w(g(x,y)\,|\,x) = \frac{\mathbb{E}(w(x,y)g(x,y)\,|\,x)}{\mathbb{E}(w(x,y)\,|\,x)}. \tag{2.35}$$

So, applied to a finite data set, ϵ_m has to be replaced by the weighted error proportions at iteration m. We will discuss the meaning of the weighted expectation in the

next section in more detail. Hence, the update in iteration m (at population level) becomes

$$F_m(x) = F_{m-1}(x) + f_m(x) \cdot \frac{1}{2} \ln \frac{1-\epsilon_m}{\epsilon_m}, \qquad w_{m+1}(x,y) = w_m(x,y) e^{-y\alpha_m f_m(x)}$$

Note that the equations in (2.33) and (2.34) refer to the *population* of a data set, rather than to concrete data points as we did in the previous sections. Therefore, this population version of AdaBoost has to be applied to data by replacing conditional expectations by weighted class proportions, as given in a terminal node of a decision tree, for example.

The population version of AdaBoost reveals another interesting property of boosting. By noting that α is chosen to minimize the objective function $J(F_{m-1}(x) + \alpha_m f_m(x))$, the following condition holds:

$$\frac{\partial J(F+\alpha f)}{\partial \alpha} = -\mathbb{E}\left(\underbrace{e^{-y(F_{m-1}(x)+\alpha_m f_m(x))}}_{w_{m+1}(x,y)} y f_m(x) \,|\, x \right) = 0$$

Since the margin $y f_m(x)$ is always $+1$ for a correct prediction and -1 for an incorrect one, the most recent weak learner $f_m(x)$ has a weighted error rate of 50% after the weights have been updated for the next iteration. An interpretation of this is, that, by reweighting the samples, the classification problem is made maximally difficult for the next weak learner [31].

RealBoost

In the previous section we have seen that the classical "Discrete" AdaBoost algorithm can be derived by minimizing the conditional expectation of the exponential error by demanding the update $\alpha_m f_m(x)$ on the ensemble $F_{m-1}(x)$ to perform a Newton-like step towards the minimum of the objective function. In this section we choose $f_m(x)$ to be the *exact* minimum of the expected error, leading to the *RealBoost* algorithm by Friedman *et al.* [11], which uses weighted probability estimates instead of "discrete" class assignments.

As before, we start from the expected exponential error conditioned on an arbitrary but fixed data point x. Suppose we currently are in iteration m and we are looking for an $f_m(x)$ that constitutes an improvement on the existing ensemble $F_{m-1}(x)$,

where we have dropped the hypothesis weight α_m. Again, the objective function $J\left(F_{m-1}(x)+f_m(x)\right)$ is given by

$$\begin{aligned} J\left(F_{m-1}(x)+f_m(x)\right) &= \mathbb{E}\left(e^{-y(F_{m-1}(x)+f_m(x))}\,|\,x\right) \\ &= \mathbb{E}\left(e^{-yF_{m-1}(x)}e^{-yf_m(x)}\,|\,x\right) \\ &= \mathbb{E}\left(e^{-yF_{m-1}(x)}\mathbf{1}_{y=1}e^{-f_m(x)}+e^{-yF_{m-1}(x)}\mathbf{1}_{y=-1}e^{f_m(x)}\,|\,x\right) \\ &= e^{-f_m(x)}\mathbb{E}\left(e^{-yF_{m-1}(x)}\mathbf{1}_{y=1}\,|\,x\right)+e^{f_m(x)}\mathbb{E}\left(e^{-yF_{m-1}(x)}\mathbf{1}_{y=-1}\,|\,x\right) \\ &= e^{-f_m(x)}\mathbb{E}_w\left(\mathbf{1}_{y=1}\,|\,x\right)+e^{f_m(x)}\mathbb{E}_w\left(\mathbf{1}_{y=-1}\,|\,x\right), \end{aligned}$$

where we have defined $w = w(x,y) = e^{-yF_{m-1}(x)}$, and $\mathbb{E}_w(\cdot\,|\,x)$ again refers to the weighted conditional expectation defined in (2.35). Setting the derivative with respect to $f_m(x)$ to zero yields

$$f_m(x) = \frac{1}{2}\ln\frac{\mathbb{E}_w\left(\mathbf{1}_{y=1}\,|\,x\right)}{\mathbb{E}_w\left(\mathbf{1}_{y=-1}\,|\,x\right)} \tag{2.36}$$

$$= \frac{1}{2}\ln\frac{P_w(y=+1\,|\,x)}{P_w(y=-1\,|\,x)} \tag{2.37}$$

We provide a proof for (2.37) in Lemma 6. In the next iteration, the ensemble becomes $F_m(x) = F_{m-1}(x)+f_m(x)$, hence the weights get updated accordingly

$$w(x,y) \leftarrow w(x,y)e^{-yf_m(x)}.$$

The entire algorithm is listed in Algorithm 5.

At a first glance, the effect of the weighted expectation and the weighted probability estimate $\mathbb{E}_w$ and P_w on the learning process and the relation between them may seem a bit vague. Before going ahead, let us stop for a second and study the population version of the RealBoost algorithm in more detail, especially its application to data. In order to get a deeper look inside the weighted probability $P_w(y=1\,|\,x)$, which a weak learner has to estimate, we revisit its original form

$$\mathbb{E}\left(e^{-yF_{m-1}(x)}\mathbf{1}_{y=1}\Big|\,x\right).$$

Defining $w(x,y) = e^{-yF_{m-1}(x)}$ and dividing by $\mathbb{E}(w(x,y)\,|\,x)$ is valid since this is equivalent to expanding the fraction in (2.36). Then, writing out the expectation yields

$$\begin{aligned} \mathbb{E}_w(\mathbf{1}_{y=1}\,|\,x) &= \frac{\mathbb{E}\left(e^{-yF_{m-1}(x)}\mathbf{1}_{y=1}\Big|\,x\right)}{\mathbb{E}\left(e^{-yF_{m-1}(x)}\Big|\,x\right)} \\ &= \frac{e^{-F_{m-1}(x)}P(y=1\,|\,x)+0\cdot P(y=-1\,|\,x)}{e^{-F_{m-1}(x)}P(y=1\,|\,x)+e^{F_{m-1}(x)}P(y=-1\,|\,x)} \\ &= \frac{P(y=1\,|\,x)}{P(y=1\,|\,x)+e^{2F_{m-1}(x)}P(y=-1\,|\,x)}, \end{aligned} \tag{2.38}$$

Algorithm 5 RealBoost

Start with weights $w_1^{(n)} \leftarrow 1/N,\ n = 1, \ldots, N$
for $m = 1$ to M **do**
  Fit the conditional distribution $P_w(y = +1 \mid x)$ by training a weak learner using weights w_m
  Compute $f_m(x) = \frac{1}{2} \ln \frac{P_w(y = +1 \mid x)}{1 - P_w(y = +1 \mid x)}$
  Update the weights: $w_{m+1} = \frac{1}{Z_m} w_m^{(n)} e^{-y_n \cdot f_m(x_n)}$
  where Z_m is a normalization factor, such that $\sum_{n=1}^{N} w_{m+1}^{(n)} = 1$
end for
Output the final classifier $\operatorname{sgn}(F(x))$, where

$$F(x) = \sum_{m=1}^{M} f_m(x)$$

which can be interpreted as the population version of a weighted probability estimate $P_w(y = 1 \mid x)$: each base classifier has to fit the function in (2.38), which has the purpose of estimating the *true* probability that a given sample has the label $y = 1$, given an input x. However, if we look at any region x in the input space where the true probability $P(y = -1 \mid x)$ is non-zero, this estimate shall be biased by the decisions of the previous classifiers. If they agree that x is dominated by the positive class (i.e. $F_{m-1}(x) > 0$), then the estimate of $P(y = 1 \mid x)$ will be mitigated exponentially fast by the negative class. In contrast, when the ensemble output is negative, the estimate is pushed towards the positive class since the influence of $P(y = -1 \mid x)$ is scaled down to 0. This effect of mediating between the true class probabilities depending on the output of previous classifiers is incorporated by the weight updating scheme of the RealBoost algorithm.

Gentle AdaBoost

In some situations, the weights $w^{(n)}$ of samples that can be predicted with large confidence may become very small. That is, the probability estimate of a weak learner $P(y = 1 \mid x)$ gets close to 1 or close to 0. In such extreme cases, that may occur when the training error gets zero, for instance, the log odd ratio update

$$f_m(x) = \frac{1}{2} \ln \frac{P_w(y = 1 \mid x)}{1 - P_w(y = 1 \mid x)} \tag{2.39}$$

to the ensemble may cause numerical problems since

$$\begin{aligned}\lim_{P_w \to 0} \left(\frac{1}{2} \ln \frac{P_w}{1-P_w} \right) &= -\infty, \text{ and} \\ \lim_{P_w \to 1} \left(\frac{1}{2} \ln \frac{P_w}{1-P_w} \right) &= \infty,\end{aligned}$$

which can result in very large updates. To overcome these problems, one solution would be to shape the update more conservatively, i.e. instead of jumping to the exact minimum of the expected exponential loss, to do just a smaller step in its direction. For this Newton stepping can be applied. Supposing an initial guess x_0 of the minimum of an objective function $J(x)$, the sequence (x_t) of Newton steps is given by

$$x_{t+1} = x_t - \frac{J'(x_t)}{J''(x_t)}, \; t = 0, \ldots \tag{2.40}$$

Applied to the exponential objective function the first and second derivative with respect to $f_m(x)$ around $f_m(x) = 0$ are given by

$$\left. \frac{\partial J(F_{m-1} + f_m(x))}{\partial f_m(x)} \right|_{f_m(x)=0} = -\mathbb{E}\left(y e^{-yF_{m-1}(x)} \,|\, x \right) \tag{2.41}$$

$$\left. \frac{\partial^2 J(F_{m-1} + f_m(x))}{\partial f_m(x)^2} \right|_{f_m(x)=0} = \mathbb{E}\left(e^{-yF_{m-1}(x)} \,|\, x \right). \tag{2.42}$$

Plugging (2.41) and (2.42) into (2.40) yields

$$F_m(x) = F_{m-1} + \frac{\mathbb{E}\left(y e^{-yF_{m-1}(x)} \,|\, x \right)}{\mathbb{E}\left(e^{-yF_{m-1}(x)} \,|\, x \right)},$$

which coincides with the definition of the weighted conditional expectation (2.35). Hence, by defining $w(x, y) = e^{-yF_{m-1}(x)}$, the update results in

$$\begin{aligned} f_m(x) &= \frac{\mathbb{E}\left(y e^{-yF_{m-1}(x)} \,|\, x \right)}{\mathbb{E}\left(e^{-yF_{m-1}(x)} \,|\, x \right)} \\ &= \mathbb{E}_w(y \,|\, x) \\ &= 1 \cdot P_w(y = 1 \,|\, x) + (-1) \cdot P_w(y = -1 \,|\, x) \\ &= 2 \cdot P_w(y = 1 \,|\, x) - 1. \end{aligned}$$

The resulting algorithm is known as *GentleBoost* and was first proposed by Friedman *et al.* [11]. Pseudo code for GentleBoost can be found in Algorithm 6.

The update to the ensemble now falls into $[-1, 1]$ and the numerical problem described above is obsolete. Since the output of each $f_m(x)$ is bounded, either the ensemble output is. Dividing $F(x)$ by M, the number of iterations, we get an ensemble output that lies in $[-1, 1]$.

Algorithm 6 GentleBoost

Start with weights $w_1^{(n)} \leftarrow 1/N$, $n = 1, \ldots, N$ and $F(x) = 0$
for $m = 1$ to M **do**
Fit the conditional distribution $P_w(y = 1 \mid x)$ by training a weak learner using weights w_m
Compute: $f_m(x) = 2 \cdot P_w(y = 1 \mid x) - 1$
Update the weights: $w_{m+1}^{(n)} = \frac{1}{Z_m} w_m^{(n)} e^{-y_n f_m(x_n)}$
where Z_m is a normalization factor, such that $\sum_{n=1}^{N} w_{m+1}^{(n)} = 1$
end for
Output the final classifier $\operatorname{sgn}(F(x))$, where

$$F(x) = \sum_{m=1}^{M} f_m(x)$$

In Chapter 3 we present evidence that GentleBoost performs almost equally to RealBoost, although the more "conservative" update would suggest far slower convergence, if at all convergence to the same stationary point. We can theoretically assess this phenomenon by comparing the GentleBoost update to the update we get if we choose the expected squared error loss,

$$\mathbb{E}\left((y - F(x))^2 \mid x\right). \tag{2.43}$$

Setting the derivative to zero yields

$$\begin{aligned} \frac{\partial \mathbb{E}\left((y - F(x))^2 \mid x\right)}{\partial F(x)} &= \mathbb{E}\left(2(y - F(x)) \mid x\right) \\ &= 2 \cdot \mathbb{E}(y \mid x) - 2 \cdot \mathbb{E}(F(x) \mid x) \\ F(x) &= \mathbb{E}(y \mid x) \\ &= 2 \cdot P(y = 1 \mid x) - 1. \end{aligned} \tag{2.44}$$

Hence we see that performing Newton steps on an exponential loss is equivalent to employing the population minimizer of the squared error loss, which is more robust in noisy settings since misclassified patterns are not penalized as much as the exponential loss does. However, since we still use the exponential for updating the weights, patterns with large positive margin will not be penalized, as the "pure" squared loss would. So the GentleBoost algorithm can be interpreted as combining two pleasant features of both approaches.

LogitBoost

As we have shown in the previous sections, by applying the AdaBoost algorithm or one of its modifications, we get to an approximate logistic regression model. However, as we have studied the effects of using different loss functions, charging an exponential loss may result in high sensitivity to noise and overlapping class distributions. As already pointed out, the major reason why boosting prefers an exponential error, despite its shortcomings, is computational, since it leads to the simple reweighting update scheme. Its similarity to logistic regression was first discovered five years after AdaBoost's inception.

Another way of obtaining an approximate logistic regression model is, instead of minimizing the exponential loss, directly maximizing the log-likelihood of the logistic regression model in (2.29). We have seen that these two losses have the same population minimizers since the log-likelihood is maximized at the true probabilities $P(y=1\,|\,x)$, which define the logit $F(x)$ in (2.30).

Assuming the logistic regression model

$$P(y=1\,|\,x)=\frac{1}{1+e^{-2F(x)}}, \tag{2.45}$$

which we define as $p(x)$, the binomial log-likelihood is

$$\begin{aligned} l\left(y^*,p(x)\right) &= \ln\left(p(x)^{y^*}\left(1-p(x)\right)^{1-y^*}\right) \\ &= y^*\ln p(x)+(1-y^*)\ln\left(1-p(x)\right) \\ &= -\ln\left(1+e^{-2yF(x)}\right). \end{aligned}$$

We additionally write $y^*=\frac{y+1}{2}$ to keep the notation uncluttered. In an arbitrary iteration m we are seeking an improved estimate of $F_{m-1}(x)$:

$$J\left(F_{m-1}(x)+f_m(x)\right)=\mathbb{E}\left(-\ln\left(1+e^{-2y\left(F_{m-1}(x)+f_m(x)\right)}\right)\middle|\,x\right) \tag{2.46}$$

Since we have no closed form solution for maximizing (2.46), we again have to employ approximate optimization techniques such as Newton stepping. The first and second derivatives at $f_m(x)=0$ are

$$\begin{aligned} \left.\frac{\partial J\left(F_{m-1}(x)+f_m(x)\right)}{\partial f_m(x)}\right|_{f_m(x)=0} &= 2\,\mathbb{E}\left(y^*-p(x)\middle|\,x\right) \\ \left.\frac{\partial^2 J\left(F_{m-1}(x)+f_m(x)\right)}{\partial f_m(x)^2}\right|_{f_m(x)=0} &= 4\,\mathbb{E}\left(p(x)\left(1-p(x)\right)\middle|\,x\right). \end{aligned}$$

Algorithm 7 LogitBoost

Start with $F(x) = 0$ and probability estimates $p_1^{(n)} = 1/2$.
for $m = 1$ to M **do**
Compute:

$$\begin{aligned} z^{(n)} &= \frac{y_n^* - p_m^{(n)}}{p_m^{(n)}(1 - p_m^{(n)})}, \qquad \text{where } y^* = \frac{y+1}{2} \\ w_m^{(n)} &= p_m^{(n)}(1 - p_m^{(n)}). \end{aligned}$$

Fit the function $f_m(x) \mapsto z$ using weights w_m.
Update:

$$F_m(x) = F_{m-1}(x) + \frac{1}{2} f_m(x), \qquad p_{m+1}^{(n)} = \frac{e^{F_m(x_n)}}{e^{F_m(x_n)} + e^{-F_m(x_n)}}$$

end for
Output the final classifier $\operatorname{sgn}(F(x))$, where

$$F(x) = \sum_{m=1}^{M} f_m(x)$$

Hence the Newton update results in

$$\begin{aligned} F_m(x) &= F_{m-1}(x) + \frac{1}{2} \frac{\mathbb{E}\left(y^* - p(x) \middle| x\right)}{\mathbb{E}\left(p(x)\left(1 - p(x)\right) \middle| x\right)} \\ &= F_{m-1}(x) + \frac{1}{2} \mathbb{E}_w \left(\frac{y^* - p(x)}{p(x)\left(1 - p(x)\right)} \middle| x \right), \end{aligned}$$

where we have defined the weights $w(x) = p(x)\left(1 - p(x)\right)$. In an implementation on data the weighted conditional expectation $\mathbb{E}_w(\cdot \mid x)$ has to be replaced by some regression method. For this, regression trees [47] or neural networks can be applied. If we rely on simple classification methods, such as decision stumps, the regression step reduces to a weighted average over samples in each of the terminal nodes. A complete listing of LogitBoost which has been proposed by Friedman *et al.* [11] is shown in Algorithm 7.

2.3. Cost-sensitive Boosting

The reweighting strategy of AdaBoost intends to increase the weight of misclassified samples and to decrease the weight of correctly classified ones, such that the weak learner of the next iteration is forced to concentrate more on the samples that are "harder" to predict. However, the weight updates of different classes are treated equally, i.e. the weights of different class samples are increased/decreased by the same update ratio. In presence of class imbalance, as pointed out in Chapter 1, the minority class often perishes due to the fact that classifier learning pursues minimization of the overall misclassification error. To overcome this undesired effect one wishes to endow the boosting strategy with means for distinguishing between classes, such that it is able to compensate for the bias towards the prevalent class. A natural way for accomplishing this is to associate misclassification costs to classes (or individual samples), that represent the importance of correctly classifying this particular class (or sample). In this section, we review some of the most important developments in cost-sensitive boosting strategies.

CSB0, CSB1 and CSB2

The earliest attempts to make the original AdaBoost procedure sensitive to costs assigned to single patterns or classes go back to Ting and Zheng [36, 34], who propose their CSB (short for **C**ost-**S**ensitive **B**oosting) algorithms in 1998. They introduce three variations of AdaBoost, which directly affect its weight update step in (2.11) and leave the rest of the procedure unsolicited. They name their modifications *CSB0*, *CSB1* and *CSB2*.

The principle idea of cost-sensitive boosting is to asymmetrically penalize misclassifications of one class by increasing the weights of the patterns belonging to this class stronger than the weights of the patterns belonging to the other class and less decreasing the weights of correct predictions of this class. The CSB0 modification is obtained by replacing the weight update formula in (2.11) of Algorithm 3 by

$$w_{m+1}^{(n)} = \frac{c^{\mathbf{1}_{y_n=1}} w_m^{(n)}}{Z_m}, \tag{2.47}$$

where c is a parameter specifying the costs of predicting a sample of the positive class to be negative and Z_m is a normalization factor chosen such that the weights sum up to 1. The significant difference to the original weight update is that (2.47) does not take into account the classification result of the weak learner, which results in monotonically increasing weights of the positive class, and monotonically decreasing

weights of the negative class, respectively. Especially, when large biased base learners are used, such as decision stumps, this results in a trivial classifier that always predicts the positive class.

The second variant, CSB1, takes the classification result of the weak learner into account but abstains from the α coefficients, such that the weight update becomes

$$w_{m+1}^{(n)} = \frac{1}{Z_m} c^{\mathbf{1}_{y_n=1}} w_m^{(n)} e^{-y_n f_m(x_n)}. \tag{2.48}$$

Finally, CSB2 uses the α_m, which are computed as in eq. (2.10), and its weight updates are defined as

$$w_{m+1}^{(n)} = \frac{1}{Z_m} c^{\mathbf{1}_{y_n=1}} w_m^{(n)} e^{-y_n \alpha_m f_m(x_n)}. \tag{2.49}$$

Note that (2.49) reduces to the original AdaBoost when costs are defined uniformly, i.e. $c = 1$.

The authors of [36, 34] do not motivate or theoretically derive their choices for the weight update formulas but they only provide an empirical study of the performances of their modifications in comparison to original AdaBoost. Considering that one way of deriving the AdaBoost algorithm is strongly driven by the weight updates as upper error bounds, we think that this heuristic approach to cost-sensitive boosting does not lead to optimal solutions. We expect modifications in the weight updates to lead to modified formulas for the α coefficients, which has been shown by Sun *et al.* [21]. We therefore discard the CSB modifications without further consideration.

AdaCost

Another approach towards cost-sensitive boosting algorithms has been published in 1999 by Fan *et al.* [30]. The principal idea of *AdaCost* is to include a cost factor in the exponent of the weight update instead of a constant factor outside the exponent, as the CSB algorithms do. In AdaCost, the weight update formula is replaced by

$$w_{m+1}^{(n)} = \frac{1}{Z_m} w_m^{(n)} e^{-\alpha_m f_m(x_n) \beta(n)}, \tag{2.50}$$

where $\beta(n) \in [-1, 1]$ is called a *cost adjustment function*. The authors recommend their definition of β to be

$$\beta(n) = \begin{cases} 0.5c_n + 0.5 & \text{if } y_n f_m(x_n) < 0 \\ -0.5c_n + 0.5 & \text{if } y_n f_m(x_n) > 0 \end{cases}, \tag{2.51}$$

Algorithm 8 AdaCost

Initalize costs $c_n \in [0,1],\ n = 1,\ldots,N$
Start with weights $w_1^{(n)} \leftarrow c_n / \sum_{n=1}^{N} c_n$
for $m = 1$ to M **do**
Fit the classifier $f_m(x) :\ \mathcal{X} \mapsto [-1,+1]$ using weights w_m
Compute

$$\alpha_m = \frac{1}{2}\ln\frac{1+r}{1-r}, \qquad r = \sum_{n=1}^{N} w_m^{(n)} y_n f_m(x_n)\beta(n) \tag{2.53}$$

$$\beta(n) = \begin{cases} 0.5c_n + 0.5 & \text{if } y_n f_m(x_n) < 0 \\ -0.5c_n + 0.5 & \text{if } y_n f_m(x_n) > 0 \end{cases} \tag{2.54}$$

Update weights:

$$w_{m+1}^{(n)} \leftarrow w_m^{(n)} \cdot \exp\left(-\alpha_m y_n f_m(x_n)\beta(n)\right) / Z_m \tag{2.55}$$

with normalization factor Z_m, such that $\sum_{n=1}^{N} w_{m+1}^{(n)} = 1$
end for
Output the final classifier $\operatorname{sgn}(F(x))$
with

$$F(x) = \sum_{m=1}^{M} \alpha_m f_m(x)$$

where $c_n \in [0,1]$ denotes the cost of misclassifying the n-th example. The authors motivate their setting of β as follows. For an instance with higher misclassification cost, the cost adjustment should increase its weight more heavily in case of misclassification than it should decrease it in case of correct classification. Following the approach of minimizing the normalization factor Z_m in each boosting round, Fan *et al.* [30] provide a choice for α_m as

$$\alpha_m = \frac{1}{2}\ln\frac{1+r_m}{1-r_m}, \qquad \text{where } r_m = \sum_{n=1}^{N} w_m^{(n)} y_n f_m(x_n)\beta(n), \tag{2.52}$$

which leads to Algorithm 8.

Although the AdaCost algorithm is often referred to in literature the choice of the cost adjustment function β of the AdaCost algorithm seems somehow arbitrary and empirical studies [21, 34] have shown, as we will confirm in Chapter 3, that AdaCost

only achieves poor performance improvements. We think one reason for this is that if costs c_n are set close to 1 for one class correct predictions of this class are not further rewarded since β gets close to zero, and hence the training process gets stuck at a margin of $yF(x) = 0$, which implies indifference of the classifier $F(x)$. Furthermore, at $yF(x) = 0$, for any choice of c_n, the weights will be updated uniformly for both classes, which neutralizes the effect of costs. Additionally, as Sun *et al.* [21] remark, there is no setting of costs c_n for which AdaCost reduces to the original AdaBoost algorithm.

AdaC1, AdaC2 and AdaC3

Sun *et al.* [21] discuss several strategies for feeding cost items into the weight update formula of AdaBoost in order to bias its reweighting update scheme towards the minority class. They consider three ways of how to modify the weight updates: inside the exponent, outside the exponent and both inside and outside the exponent. The resulting boosting modifications are known as *AdaC1*, *AdaC2* and *AdaC3* and their corresponding mixing coefficients α_m can be induced by minimizing the normalization factor Z_m in each boosting round, as the original AdaBoost algorithm has been derived in Section 2.1.

For AdaC1, the weight update rule is modified as follows:

$$w_{m+1}^{(n)} = \frac{1}{Z_m} w_m^{(n)} e^{-c_n y_n \alpha_m f_m(x_n)}, \tag{2.56}$$

i.e. a constant cost factor c_n is inserted into the exponent of the weight updates for each pattern class. Usually, in the class imbalance case, c_n is set to $c^{\mathbf{1}_{y_n=1}}$ for all n. As before, the weights after the M-th boosting round are given by

$$w_{M+1}^{(n)} = \frac{e^{-c_n y_n \sum_{m=1}^{M} \alpha_m f_m(x_n)}}{\prod_{m=1}^{M} Z_m} = \frac{e^{-c_n y_n F_M(x_n)}}{\prod_{m=1}^{M} Z_m}, \tag{2.57}$$

and for $c \geq 1$, the overall training error is bounded by

$$\begin{aligned} \sum_{n=1}^{N} \mathbf{1}_{y_n \neq F_M(x_n)} &\leq \sum_{n=1}^{N} e^{-c_n y_n F_M(x_n)} \\ &= \sum_{n=1}^{N} \left(\prod_{m=1}^{M} Z_m \right) w_{M+1}^{(n)} \\ &= \prod_{m=1}^{M} Z_m. \end{aligned}$$

Splitting Z_m into positive and negative predictions yields

$$Z_m = \sum_{n=1}^{N} w_m^{(n)} e^{-c_n y_n f_m(x_n)} \leq \sum_{n=1}^{N} w_m^{(n)} \left(\frac{1}{2} \left(c_n + c_n y_n f_m(x_n) \right) e^{-\alpha_m} + \frac{1}{2} \left(c_n - c_n y_n f_m(x_n) \right) e^{\alpha_m} \right),$$

which, by setting the derivative with respect to α_m to zero, results in

$$\alpha_m = \frac{1}{2} \ln \frac{\sum_{n=1}^{N} w_m^{(n)} c_n + \sum_{n=1}^{N} w_m^{(n)} c_n f_m(x_n) y_n}{\sum_{n=1}^{N} w_m^{(n)} c_n - \sum_{n=1}^{N} w_m^{(n)} c_n f_m(x_n) y_n}.$$

Algorithm 9 AdaC1

Start with weights $w_m^{(n)} \leftarrow 1/N,\ n = 1, \ldots, N$
for $m = 1$ to M **do**
Fit the classifier $f_m(x) \in \{-1, +1\}$ using weights w_m
Compute

$$\alpha_m = \frac{1}{2} \ln \frac{\sum_{n=1}^{N} w_m^{(n)} c_n + \sum_{n=1}^{N} w_n c_n f_m(x_n) y_n}{\sum_{n=1}^{N} w_m^{(n)} c_n - \sum_{n=1}^{N} w_m^{(n)} c_n f_m(x_n) y_n}. \tag{2.58}$$

Update the weights $w_{m+1}^{(n)} = \frac{1}{Z_m} w_m^{(n)} e^{-c_n y_n \alpha_m f_m(x_n)}$
where Z_m is a normalization factor, such that $\sum_{n=1}^{N} w_{m+1}^{(n)} = 1$.
end for
Output the final classifier $\operatorname{sgn}(F(x))$ where

$$F(x) = \sum_{m=1}^{M} \alpha_m f_m(x)$$

Algorithm 9 summarizes the whole description of AdaC1. The AdaC2 and AdaC3 modifications can be obtained the same way. The weight updates are

- AdaC2: $w_{m+1}^{(n)} = \frac{1}{Z_m} c_n w_m^{(n)} e^{-\alpha_m f_m(x_n) y_n}$
- AdaC3: $w_{m+1}^{(n)} = \frac{1}{Z_m} c_n w_m^{(n)} e^{-c_n \alpha_m f_m(x_n) y_n}$.

Since the line of proof is exactly the same for AdaC2 and AdaC3 we waive the derivation of the corresponding α-values and leave it at referring to Algorithms 10 and 11.

Algorithm 10 AdaC2

Start with weights $w_m^{(n)} \leftarrow 1/N,\ n = 1, \ldots, N$
for $m = 1$ to M **do**
Fit the classifier $f_m(x) \in \{-1, +1\}$ using weights w_m

$$\alpha_m = \frac{1}{2} \ln \frac{\sum_{n=1}^{N} \mathbf{1}_{y_n = f_m(x_n)} w_m^{(n)} c_n}{\sum_{n=1}^{N} \mathbf{1}_{y_n \neq f_m(x_n)} w_m^{(n)} c_n}$$

Update the weights: $w_{m+1}^{(n)} = \frac{1}{Z_m} w_m^{(n)} c_n e^{-y_n \alpha_m f_m(x_n)}$
where Z_m is a normalization factor, such that $\sum_n^N w_{m+1}^{(n)} = 1$.
end for
Output the final classifier $\operatorname{sgn}(F(x))$ where

$$F(x) = \sum_{m=1}^{M} \alpha_m f_m(x)$$

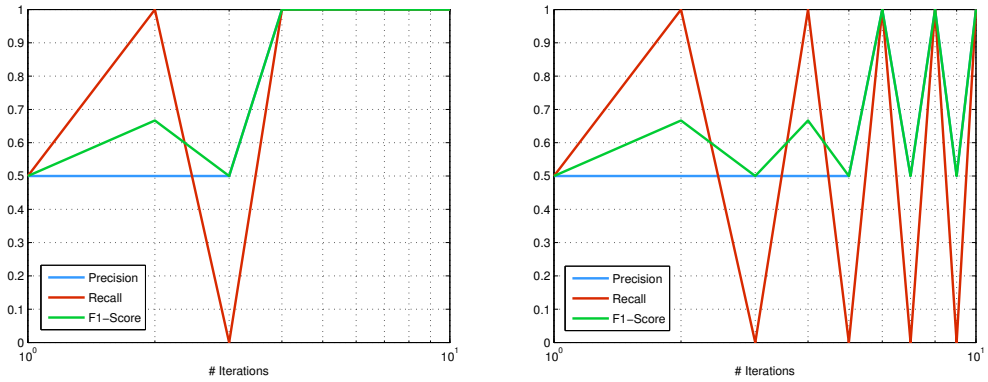
The AdaC1-3 algorithms have been empirically evaluated by several authors [21, 76, 39]. Masnadi-Shirazi and Vasconcelos [39] report on AdaC1 being unstable in some situations but they do not particularize or analyze their findings. In Chapter 3 we empirically confirm the instability of AdaC1. Indeed, we observe that AdaC1 starts oscillating when costs are chosen adversely.

Let us try to explain this behavior by means of a trivial toy example. Consider a 1-dimensional classification problem consisting of 3 data points, $Z = \{(x_1, y_1 = -1), (x_2, y_2 = +1), (x_3, y_3 = -1)\}$, with $x_1 < x_2 < x_3$ and cost assignments $c_1 = c_3 = 1$ and $c_2 = c \gg 1$. Suppose we have decision stumps available as base classifiers (i.e. a threshold for x in each iteration). Discrete AdaBoost is able to solve the problem after 4 iterations. Let us qualitatively study what happens in AdaC1 during the first iterations.

1. Suppose the threshold found in the first iteration lies between x_1 and x_2, so x_1 and x_2 are classified correctly and x_3 is misclassified. According to (2.58), $\alpha_1 > 0$ and the weights get updated

$$w_2^{(1)} = 1/3\, e^{-\alpha_1},\ w_2^{(2)} = 1/3\, e^{-c\alpha_1} \text{ and } w_2^{(3)} = 1/3\, e^{\alpha_1}.$$

2. Since c is large the weight updates cause $w_2^{(2)}$ to be close to 0, whereas $w_2^{(1)}$ is not scaled down as much, and $w_2^{(3)}$ is increased due to misclassification. Therefore, in the second iteration, if $w_2^{(1)} + w_2^{(3)} > w_2^{(2)}$, the base learner is



(a) *AdaBoost* **(b)** *AdaC1*

Figure 2.3.: *(a) AdaBoost and (b) AdaC1 run on the toy example with $c = 3$ over 10 iterations: AdaBoost solves the problem after 4 iterations; AdaC1 shows oscillations.*

likely to constantly predict the negative class (i.e. $f_2(x) = -1$), since there is no threshold achieving lower weighted error, which causes α_2 to turn negative and the weight updates

$$w_3^{(1)} = w_2^{(1)} e^{\alpha_2},\ w_3^{(2)} = w_2^{(2)} e^{-c\alpha_2} \text{ and } w_3^{(3)} = w_2^{(3)} e^{\alpha_2},$$

such that in the next iteration, due to c, $w_3^{(2)}$ becomes very large and the base learner is forced to correctly classify x_2.

It is traceable that the exponential influence of the cost factor c results in strongly oscillating weights of one class, which induces oscillating predictive behavior, since strong penalties of false-negative predictions are compensated by strong rewards of true-positives. In order to illustrate this effect we run AdaC1 and Discrete AdaBoost over 10 iterations on the above toy example, having set $c = 3$. The results are shown in Figure 2.3. As can be seen, AdaBoost succeeds in solving the problem after 4 iterations, whereas AdaC1 does not converge, thus this approach seems not to be promising.

Asymmetric Logistic Regression

In the previous sections we considered cost-sensitive extensions to the original AdaBoost algorithm. The AdaC1-3 algorithms are obtained by inserting constant cost factors at different positions in the weight update and by minimizing an upper bound on the training error like we did in Section 2.1. We also provided another way to derive the AdaBoost algorithm at population level by minimizing the conditional expec-

Algorithm 11 AdaC3

Start with weights $w_n \leftarrow \frac{1}{N},\ n = 1,\ldots,N$
for $m = 1$ to M **do**
Fit the classifier $f_m(x) \in \{-1,+1\}$ using weights w_n

$$\alpha_m = \frac{1}{2}\ln\frac{\sum_{n=1}^{N} w_m^{(n)} c_n + \sum_{n=1}^{N} w_m^{(n)} c_n^2 y_n f_m(x_n)}{\sum_{n=1}^{N} w_m^{(n)} c_n - \sum_{n=1}^{N} w_m^{(n)} c_n^2 y_n f_m(x_n)}$$

Update the weights: $w_{m+1} \leftarrow \frac{1}{Z_m} w_m^{(n)} c_n e^{-c_n y_n \alpha_m f_m(x_n)}$,
where Z_m is a normalization factor such that $\sum_{n=1}^{N} w_{m+1}^{(n)} = 1$
end for
Output the final classifier $\operatorname{sgn}(\sum_{m=1}^{M} \alpha_m f_m(x))$

tation of an exponential misclassification function of the margin $yF(x)$, giving rise to the RealBoost, GentleBoost and LogitBoost procedures having several advantages over classical Discrete AdaBoost. Corresponding cost-sensitive extensions to these algorithms exist, which we want to introduce in this section.

Li *et al.* [28] introduce an asymmetric logistic regression model of the form

$$F(x) = \frac{1}{2}\ln\frac{P(y=1\,|\,x)c}{1-P(y=1\,|\,x)} = \frac{1}{2}\ln\frac{P(y=1\,|\,x)}{1-P(y=1\,|\,x)} + \frac{1}{2}\ln c. \tag{2.59}$$

The constant factor c in the numerator of the logit transform achieves a decision threshold that is $\frac{1}{2}\ln c$ smaller than it would be in the normal logistic regression model. This leads to more samples being predicted as positive to the expense of increasing false positive cases, hence the model is biased towards the positive class. It can be shown that a model of the form (2.59) can be obtained at population level by employing the asymmetric exponential loss

$$\mathbb{E}\left(c^{y^*} e^{-yF(x)}\Big|\, x\right), \tag{2.60}$$

where we have defined $y^* = \frac{y+1}{2}$. Here, the cost factor c can be interpreted in a way that a misclassification of a positive sample is c times as expensive as making an error on a negative sample. Before deriving an additive asymmetric logistic regression model from the expectation in (2.60) we have to deploy the objective function to be minimized in each iteration. For this we note that

$$J\left(F(x)\right) = \mathbb{E}\left(c^{y^*} e^{-yF(x)}\Big|\, x\right) = \mathbb{E}\left(c^{y^*} e^{-y\sum_{m=1}^{M} f_m(x)}\Big|\, x\right) = \mathbb{E}\left(\prod_{m=1}^{M} c^{y^*/M} e^{-y f_m(x)}\Big|\, x\right), \tag{2.61}$$

such that in each iteration, the asymmetric cost $c^{y^*/M}$ is assigned to the positive samples. At the end of this section, we will discuss why this step is necessary for the algorithms to work.

Cost-Sensitive RealBoost

Let us now derive a cost-sensitive version of the RealBoost algorithm. Let $k = c^{1/M}$ and suppose we want an improved estimate $F_{m-1}(x) + f_m(x)$ in the current iteration. We have to minimize the objective function with respect to $f_m(x)$ at an arbitrary but fixed x:

$$\begin{aligned} J\left(F_{m-1} + f_m\right) &= \mathbb{E}\left(k^{my^*} e^{-y(F_{m-1}(x)+f_m(x))} \middle| x\right) \\ &= \mathbb{E}\left(k^{my^*} e^{-yF_{m-1}(x)-yf_m(x)} \middle| x\right) \\ &= e^{-f_m(x)}\mathbb{E}\left(k^{my^*} e^{-yF_{m-1}(x)} \mathbf{1}_{y=1} \middle| x\right) \\ &\qquad + e^{f_m(x)}\mathbb{E}\left(k^{my^*} e^{-yF_{m-1}(x)} \mathbf{1}_{y=-1} \middle| x\right) \\ &= e^{-f_m(x)}\mathbb{E}_w\left(\mathbf{1}_{y=1} \middle| x\right) + e^{f_m(x)}\mathbb{E}_w\left(\mathbf{1}_{y=-1} \middle| x\right) \end{aligned}$$

where we have defined $w(x,y) = k^{my^*} e^{-yF_{m-1}(x)}$. Again, $\mathbb{E}_w(\cdot \,|\, x)$ refers to the conditional weighted expectation defined in (2.35). Setting the derivative to zero yields

$$f_m(x) = \frac{1}{2}\ln\frac{\mathbb{E}_w\left(\mathbf{1}_{y=1} \middle| x\right)}{\mathbb{E}_w\left(\mathbf{1}_{y=1} \middle| x\right)} = \frac{1}{2}\ln\frac{P_w(y=1\,|\,x)}{1-P_w(y=1\,|\,x)}, \tag{2.62}$$

and the weights get updated by $w(x,y) \leftarrow w(x,y)k^{y^*} e^{-yf_m(x)}$. This leads to the CSRA (**C**ost-**S**ensitive **R**eal**A**daBoost) procedure listed in Algorithm 12.

Cost-Sensitive GentleBoost

In the same way we can obtain a cost-sensitive version of the GentleBoost algorithm, by performing a Newton step on the asymmetric exponential loss in each iteration:

$$\begin{aligned} \left.\frac{\partial J(F_{m-1}+f_m(x))}{\partial f_m(x)}\right|_{f_m(x)=0} &= -\mathbb{E}\left(yk^{y^*} e^{-yF_{m-1}(x)} \,|\, x\right) \\ \left.\frac{\partial^2 J(F_{m-1}+f_m(x))}{\partial f_m(x)^2}\right|_{f_m(x)=0} &= \mathbb{E}\left(k^{y^*} e^{-yF_{m-1}(x)} \,|\, x\right). \end{aligned}$$

Hence, the Newton update becomes

$$f_m(x) = \frac{\mathbb{E}\left(yk^{y^*} e^{-yF_{m-1}(x)} \,|\, x\right)}{\mathbb{E}\left(k^{y^*} e^{-yF_{m-1}(x)} \,|\, x\right)} = \mathbb{E}_w(y\,|\,x) = 2P_w(y=1\,|\,x) - 1, \tag{2.63}$$

Algorithm 12 CSRA

Start with weights $w_1^{(n)} \leftarrow 1/N,\ n = 1, \ldots, N$.
Compute $k = c^{1/M}$, with false negative costs c.
for $m = 1$ to M **do**
Fit the conditional distribution $P_w(y = +1 \,|\, x)$ by training a weak learner using weights w_m
Compute $f_m(x) = \frac{1}{2} \ln \frac{P_w(y = +1 \,|\, x)}{1 - P_w(y = +1 \,|\, x)}$
Update the weights: $w_{m+1}^{(n)} = \frac{1}{Z_m} k^{y^*} w_m^{(n)} e^{-y_n \cdot f_m(x_n)}, \quad y^* = \frac{y+1}{2}$
where Z_m is a normalization factor, such that $\sum_{n=1}^{N} w_{m+1}^{(n)} = 1$
end for
Output the final classifier $\operatorname{sgn}(F(x))$, where

$$F(x) = \sum_{m=1}^{M} f_m(x)$$

where the weights are again updated by $w(x, y) \leftarrow w(x, y) k^{y^*} e^{-y f_m(x)}$. The complete CSGA (**C**ost-**S**ensitive **G**entle **A**daBoost) is shown in Algorithm 13.

Let us analyze the impact of introducing asymmetric costs into the exponential loss on the weighted probability estimates in more detail. According to the weighted conditional expectation in (2.35) we write, having defined $w(x, y) = k^{y^*} e^{-y F_{m-1}(x)}$,

$$\begin{aligned} P_w(y = 1 \,|\, x) = \mathbb{E}_w(\mathbf{1}_{y=1} \,|\, x) &= \frac{\mathbb{E}\left(k^{y^*} e^{-y F_{m-1}(x)} \mathbf{1}_{y=1} \middle| x\right)}{\mathbb{E}\left(k^{y^*} e^{-y F_{m-1}(x)} \middle| x\right)} \\ &= \frac{k e^{-F_{m-1}(x)} P(y = 1 \,|\, x) + 0 \cdot P(y = -1 \,|\, x)}{k e^{-F_{m-1}(x)} P(y = 1 \,|\, x) + e^{F_{m-1}(x)} P(y = -1 \,|\, x)} \\ &= \frac{P(y = 1 \,|\, x)}{P(y = 1 \,|\, x) + 1/k\, e^{2F_{m-1}(x)} P(y = -1 \,|\, x)}. \end{aligned} \tag{2.64}$$

We see that the weighted probability estimate of an asymmetric loss (i.e. its population version) is equivalent to the weighted probability estimate of the symmetric loss, except that the influence of the negative class is stemmed by a factor $1/k$. It is now obvious why a splitting of the costs over the iterations in (2.61) is necessary: Suppose we are in iteration m and according to (2.64) the base learner is biased towards the positive class. From (2.59) we know that the bias is $1/2 \ln c$, such that the model of the next iteration becomes

$$F_m(x) = F_{m-1}(x) + \frac{1}{2} \ln \frac{P_w(y = 1 \,|\, x)}{1 - P_w(y = 1 \,|\, x)} + \frac{1}{2} \ln c$$

Algorithm 13 CSGA

Start with weights $w_1^{(n)} \leftarrow 1/N,\ n = 1, \ldots, N$ and $F(x) = 0$
Compute $k = c^{1/M}$, with false negative costs c.
for $m = 1$ to M **do**
  Fit the conditional distribution $P_w(y = 1 \mid x)$ by training a weak learner using weights w_m
  Compute: $f_m(x) = 2 \cdot P_w(y = 1 \mid x) - 1$
  Update the weights: $w_{m+1}^{(n)} = \frac{1}{Z_m} k^{y^*} w_m^{(n)} e^{-y_n f_m(x_n)}$
  where Z_m is a normalization factor, such that $\sum_{n=1}^{N} w_{m+1}^{(n)} = 1$
end for
Output the final classifier $\operatorname{sgn}(F(x))$, where

$$F(x) = \sum_{m=1}^{M} f_m(x)$$

and according to the weight updates the weighted probability estimate of the next iteration is given by

$$\begin{aligned} P_w(y = 1 \mid x) &= \frac{P(y = 1 \mid x)}{P(y = 1 \mid x) + 1/k\, e^{2F_{m-1}(x) + \ln \frac{P_w(y=1 \mid x)}{P_w(y=-1 \mid x)} + \ln c} P(y = -1 \mid x)} \\ &= \frac{P(y = 1 \mid x)}{P(y = 1 \mid x) + c/k\, e^{2F_{m-1}(x) + \frac{P_w(y=1 \mid x)}{P_w(y=-1 \mid x)}} P(y = -1 \mid x)}. \end{aligned} \tag{2.65}$$

Thus, if at each iteration m the cost factor k is chosen to be a constant c, then after one iteration the biasing effect of costs will vanish. Hence, it is important for cost-sensitive boosting algorithms to work that the cost factors increase in each iteration in order buoy up the bias towards the positive class, as it is suggested by (2.61).

Cost-Sensitive LogitBoost

A corresponding cost-sensitive modification of the LogitBoost algorithm was first proposed by Li *et al.* [28]. However, we detected several flaws in their derivation of the algorithm and therefore want to present a correct version in this thesis.

From the logit in (2.59) we get the corresponding asymmetric class probabilities by inverting, i.e.

$$\begin{aligned} p(x) = P(y=1\,|\,x) &= \frac{1}{1+ce^{-2F(x)}} \\ &= \frac{1}{1+ce^{-2\sum_{m=1}^{M} f_m(x)}} \\ &= \frac{1}{1+\prod_{m=1}^{M} ke^{-2f_m(x)}}, \end{aligned}$$

where we have defined $k = c^{1/M}$, such that the expected asymmetric log-likelihood becomes

$$-\,\mathbb{E}\left(\ln\left(1+c^{y}e^{-2yF(x)}\right)\middle|\,x\right), \tag{2.66}$$

which we want to maximize by Newton steps in order to obtain a cost-sensitive variant of the LogitBoost algorithm.

$$\begin{aligned} \left.\frac{\partial J\left(F_{m-1}(x)+f_m(x)\right)}{\partial f_m(x)}\right|_{f_m(x)=0} &= 2\,\mathbb{E}\left(y^{*}-p(x)\middle|\,x\right) \\ \left.\frac{\partial^2 J\left(F_{m-1}(x)+f_m(x)\right)}{\partial f_m(x)^2}\right|_{f_m(x)=0} &= 4\,\mathbb{E}\left(p(x)\left(1-p(x)\right)\middle|\,x\right). \end{aligned}$$

As can be seen, the only thing that has to be modified to come from the original LogitBoost algorithm to a cost sensitive extension, is to alter the probability estimates in each round to

$$p_{m+1}(x) = \frac{1}{1+k^{m}e^{-2F_m(x)}}$$

This leads to the CSLB (**C**ost-**S**ensitive **L**ogit**B**oost) procedure described in Algorithm 14.

Exponential Cost Items

In the previous section we derived cost-sensitive versions of the probabilistic boosting variants RealBoost, GentleBoost and LogitBoost. In doing so, we inserted a class-dependent cost factor c into the exponential loss function, which causes the base learner to be biased towards one of the classes by giving the patterns of the respective class constantly higher weights. In this case, a cost item has been inserted as a

Algorithm 14 CSLB

Compute $k = c^{1/M}$, with false negative costs c.
Start with $F(x) = 0$ and probability estimates $p_1^{(n)} = \frac{1}{1+k}$.
for $m = 1$ to M **do**
Compute:

$$\begin{aligned} z^{(n)} &= \frac{y_n^* - p_m^{(n)}}{p_m^{(n)}(1-p_m^{(n)})}, \qquad \text{where } y^* = \frac{y+1}{2} \\ w_m^{(n)} &= p_m^{(n)}(1-p_m^{(n)}). \end{aligned}$$

Fit the function $f_m(x) \mapsto z$ using weights w_m.
Update:

$$F_m(x) = F_{m-1}(x) + \frac{1}{2} f_m(x), \qquad p_{m+1}^{(n)} = \frac{1}{1 + k^m e^{-2F_m(x_n)}}$$

end for
Output the final classifier $\operatorname{sgn}(F(x))$, where

$$F(x) = \sum_{m=1}^{M} f_m(x)$$

multiplicative term outside the exponent of the error loss. Alternatively, the cost item can also be placed in the exponent, such that the expected conditional loss becomes

$$\mathbb{E}\left(e^{-yc^{y^*}F(x)}\,|\,x\right),$$

where $y^* = \frac{y+1}{2}$ again. The effect of using an exponential cost item instead of a multiplicative one shall be discussed in the following. Starting from the premise that we are seeking an improvement on the ensemble we have learned so far, i.e. $F_{m-1}(x) + f_m(x)$, the expected exponential loss becomes

$$\mathbb{E}\left(e^{-yc^{y^*}\left(F_{m-1}(x)+f_m(x)\right)}\,|\,x\right). \tag{2.67}$$

Minimizing (2.67) with respect to $f_m(x)$ as it has been done in the previous section yields

$$f_m(x) = \frac{1}{1+c}\ln\frac{cP_w(y=1\,|\,x)}{1-P_w(y=1\,|\,x)} = \frac{1}{1+c}\ln\frac{P_w(y=1\,|\,x)}{1-P_w(y=1\,|\,x)} + \frac{1}{1+c}\ln c, \tag{2.68}$$

where we have defined the weights $w(x,y) = e^{-yc^{y^*}F_{m-1}(x)}$. Let us study the effect of using exponential costs on the weighted probability estimate of the base classifier. Starting from the weighted conditional expectation we get

$$\begin{aligned} P_w(y=1\,|\,x) = \mathbb{E}_w(\mathbf{1}_{y=1}\,|\,x) &= \frac{\mathbb{E}\left(e^{-yc^{y^*}F_{m-1}(x)}\mathbf{1}_{y=1}\,\middle|\,x\right)}{\mathbb{E}\left(e^{-yc^{y^*}F_{m-1}(x)}\,\middle|\,x\right)} \\ &= \frac{e^{-cF_{m-1}(x)}P(y=1\,|\,x) + 0\cdot P(y=-1\,|\,x)}{e^{-cF_{m-1}(x)}P(y=1\,|\,x) + e^{F_{m-1}(x)}P(y=-1\,|\,x)} \\ &= \frac{P(y=1\,|\,x)}{P(y=1\,|\,x) + e^{(1+c)F_{m-1}(x)}P(y=-1\,|\,x)} \end{aligned} \tag{2.69}$$

as the population version of the weighted probability $P_w(y=1\,|\,x)$. At a first glance, the effect of c seems to be counter-intuitive, since it strengthens the influence of the negative class on the training process and hence stems the (relative) influence of the positive class. Indeed, if a boosting ensemble is trained using weights as defined above and the update at iteration m is given by (2.68), the base learners are biased towards the negative class instead of the positive one. However, considering the ensemble output, the bias of the base learner is compensated by the constant term $\frac{1}{1+c}\ln c$ in (2.68). Furthermore, with increasing costs c the entire update $f_m(x)$ will vanish, such that the ensemble output converges to 0, which corresponds to a classifier making random guesses. Masnadi-Shirazi and Vasconcelos [39] yet report on improved performance of such a model in dealing with rare classes. Against the background of our theoretical investigations, we can explain this by means of the

insensitivity of a base classifier to data. In these terms, the classifier is expected to pay less attention to the increased influence of the negative class, which causes the log odd ratio in (2.68) to be smaller than the absolute bias $\frac{1}{1+c}\ln c$. However, with increasing costs c this effect will vanish and the ensemble output converges to 0.

2.4. Boosting with Imbalanced Classes

In the previous section we described cost-sensitive boosting approaches by introducing cost items for particular patterns (i.e. all patterns of the rare class) with the goal of compensating for the bias of the learner towards the prevalent class induced by its numerical dominance. In this section, we present an alternative approach for tackling the imbalance problem from the algorithmic and the data level, that are reported in literature.

RareBoost

As described in Chapter 1, a classifier achieves high recall by learning a powerful model for distinguishing false-negative (FN) from true-negative (TN) and high precision by distinguishing false-positive (FP) from true-positive (TP) cases. In general, precision and recall are conflicting goals and, as we have seen, optimizing the overall misclassification error (i.e. accuracy) does not necessarily lead to desirable precision and recall values. Joshi *et al.* [32] approach this problem by giving different treatment to FPs and FNs. In particular, the idea is to learn different α_m coefficients in each iteration m for positive and negative predictions. We denote these as α_m^+ for positive predictions, (i.e. $f(x) > 0$) and as α_m^- for the negative ones ($f(x) < 0$). Based on its classification result, a hypothesis $f_m(x)$ will be weighted by the corresponding α_m, such that the final decision of the ensemble is obtained by

$$F(x) = \sum_{m:f_m(x)\geq 0} \alpha_m^+ f_m(x) + \sum_{m:f_m(x)<0} \alpha_m^- f_m(x). \tag{2.70}$$

The α values can be obtained as follows. By recursively expanding the weight updates over all iterations, we get

$$\begin{aligned} w_{M+1}^{(n)} &= \frac{1}{N\prod_{m=1}^M Z_m} \exp\left(-\left(\sum_{m:f_m(x_n)\geq 0} y_n \alpha_m^+ f_m(x_n) + \sum_{m:f_m(x_n)<0} y_n \alpha_m^- f_m(x)\right)\right) \\ &= \frac{\exp\left(y_n F(x_n)\right)}{N\prod_{m=1}^M Z_m}. \end{aligned}$$

Following the proof of Schapire and Singer [7], the training error is minimized by greedily minimizing Z_m, which in this case is the sum of weights of the positive and the negative predictions after iteration m, i.e. $Z_m = Z_m^+ + Z_m^-$, where

$$Z_m^+ = \sum_{m:f_m(x_n)\geq 0} y_n \alpha_m^+ f_m(x_n), \qquad Z_m^- = \sum_{m:f_m(x_n)<0} y_n \alpha_m^- f_m(x).$$

Each of the Z_m can be minimized by setting the derivative with respect to $\alpha_m^\pm$ to zero, yielding

$$\alpha_m^+ = \frac{1}{2}\ln\frac{TP_m}{FP_m}, \qquad \text{where} \begin{cases} TP_m = \sum_{n:f_m(x_n)=1,y_n=1} w_m^{(n)} f_m(x_n) \\ FP_m = \sum_{n:f_m(x_n)=1,y_n=-1} w_m^{(n)} f_m(x_n) \end{cases}$$

$$\alpha_m^- = \frac{1}{2}\ln\frac{TN_m}{FN_m}, \qquad \text{where} \begin{cases} TN_m = \sum_{n:f_m(x_n)=-1,y_n=-1} w_m^{(n)} f_m(x_n) \\ FN_m = \sum_{n:f_m(x_n)=-1,y_n=1} w_m^{(n)} f_m(x_n) \end{cases}$$

The resulting algorithm is known as *RareBoost* and shown in Algorithm 15. The principal advantage of RareBoost over classical AdaBoost is the distinction between positive and negative predictions. Consider, for example, an imbalanced setting, where the negative prevalent class achieves high accuracy, whereas the minority class is predicted only infrequently. In such a scenario, we expect to get a very large amount of TN cases, such that the α_m^- for negative predictions get large. In this case, an FN sample, that belongs to the rare class but is misclassified, experiences a large increase of its weights, such that the learner of the next iteration will focus more on that example. On the other hand, since the positive predictions are rare, we expect the base classifier to produce only few TP cases, such that a misclassified example of the negative (prevalent) class will gain only little increase of its weight since the corresponding α_m^+ value is small. In the original AdaBoost algorithm these two cases are treated equally, so FP and FN cases get updated by the same ratio.

Although the RareBoost algorithm indeed often performs significantly better than AdaBoost, there are some points of criticism that have been reported in literature. Sun *et al.* [21], for instance, remark that the RareBoost update scheme is only able to scale down the training error if the base learner achieves performance with

$$TP > FP \qquad \textit{and} \qquad TN > FN$$

and collapses if this constraint is not satisfied. They argue that, especially in presence of class imbalance, demanding $TP > FP$ is a too strong condition since it is equivalent to demanding a precision value greater 0.5. Imbalanced problems, however, were characterized by *both* poor recall and precision. We disagree with this argument,

since our experiments have shown that mostly recall suffers from imbalance, whereas precision is relatively high in most cases. By inspecting Algorithm 15, one can see that RareBoost indeed collapses if the condition $TP > FP$ is violated, while $TN > FN$ holds. In this case, TP predictions are penalized by increasing their weights (since α_m^+ is negative), while TN predictions are rewarded (since α_m^- is positive). However, we can show that as long as the base learner minimizes its weighted error rate this case is impossible to occur, which rebuts the above arguments. For showing this, we introduce the following lemma.

Lemma 1 *Assume a binary classification problem, i.e.* $y \in \{-1, +1\}$. *Let* $F(x)$ *be a classifier producing errors* $FP > TP$ *and* $TN > FN$. *Then the trivial classifier* $F'(x) \equiv -1$ *achieves a lower misclassification error than* $F(x)$*:*

$$\sum_{n=1}^{N} \mathbf{1}_{F(x_n) \neq y_n} > \sum_{n=1}^{N} \mathbf{1}_{F'(x_n) \neq y_n} \tag{2.71}$$

Proof The trivial cases of equality of TP and FP as well as TN and FN, respectively, can be ruled out since we demand the classifier to be better than random guessing. For $F'(x) \equiv -1$ the following holds:

$$TP' = FP' = 0,\ TN' = TN + FP,\ FN' = TP + FN.$$

We assume that $F(x)$ achieves a lower misclassification error than $F'(x)$. Then

$$\begin{aligned} \sum_{n=1}^{N} \mathbf{1}_{F(x_n) \neq y_n} &< \sum_{n=1}^{N} \mathbf{1}_{F'(x_n) \neq y_n} \\ FP + FN &< FP' + FN' \\ FP + FN &< TP + FN \\ \Longleftrightarrow FP &< TP, \end{aligned}$$

which contradicts the assumption $FP > TP$.

□

Note that we postulate $TN > FN$ for $F(x)$ in Lemma 1. Although Lemma 1 would also hold without this demand, it is a reasonable assumption since a classifier $F(x)$ with $FP > TP$ and $FN > TN$ can easily be inverted into $-F(x)$, which then has $TP > FP$ and $TN > FN$.

Lemma 1 tells us that whenever we have trained a classifier which produces more errors than correct predictions regarding the one of the two classes, we can always do better when we constantly predict the other class. The proof for the case $FN > TN$ and $TP > FP$ is analogous. This means, as long as the misclassification error of the base classifier we use is upper-bounded by the trivial classifier, we can show that the situation described above cannot occur. We state this in

Proposition 2 *Let $F(x)$ be a classifier that minimizes the overall misclassification rate, such that*

$$F(\mathbf{x}) = \arg\min_{F} \frac{1}{N} \sum_{n=1}^{N} \mathbf{1}_{F(x_n) \neq y_n}, \tag{2.72}$$

Then the following holds for its performance measures TP, FP, TN and FN:

$$(TP > FP \wedge TN > FN) \vee (FP > TP \wedge FN > TN), \tag{2.73}$$

where we have omitted the cases of equality.

Proof We prove our proposition by refuting the remaining cases

$$(TP < FP \wedge TN > FN) \text{ and } (FP < TP \wedge FN > TN).$$

1. $\neg(TP < FP \wedge TN > FN)$: We assume $TP < FP \wedge TN > FN$. Then the trivial classifier $F'(x) \equiv -1$ achieves a lower misclassification rate than $F(x)$, which follows from Lemma 1, such that $F(x)$ does not satisfy (2.72).

2. $\neg(FP < TP \wedge FN > TN)$: analogous to 1.

□

From Proposition 2 we can conclude that the particular situation in which the RareBoost algorithm is supposed to collapse cannot occur as long as the base classifier we use minimizes its training error and is able to act as a trivial classifier which constantly predicts one of the two classes. Thus we have disproved the above statements by Sun *et al.*

Nevertheless, we can also show that RareBoost is *not* a solution of the imbalance problem, since it can be interpreted as an approximation to the RealBoost algorithm. In fact, RareBoost is even equivalent to RealBoost, if decision stumps are used as base learners. We state this in

Proposition 3 *The RareBoost algorithm is equivalent to the RealBoost algorithm if decision stumps are used as base classifiers.*

Proof We prove the identity of RareBoost and RealBoost in the case where the weak learner is a decision stump with $f_m \in \{-1, 1\}$. From the RareBoost update scheme for positive predictions ($f_m(x) \geq 0$) it follows that

$$\begin{aligned}
\alpha_m^+ &= \frac{1}{2}\ln(TP_m/FP_m) \\
&= \frac{1}{2}\ln\frac{TP_m/\sum_{n:f_m(x_n)\geq 0} w_m^{(n)} f_m(x_n)}{FP_m/\sum_{n:f_m(x_n)\geq 0} w_m^{(n)} f_m(x_n)} \\
&= \frac{1}{2}\ln\frac{P_w(TP\,|\,x)}{P_w(FP\,|\,x)} \\
&= \frac{1}{2}\ln\frac{P_w(y=1\,|\,x)}{P_w(y=-1\,|\,x)} = \frac{1}{2}\ln\frac{P_w(y=1\,|\,x)}{1-P_w(y=1\,|\,x)} \\
&=: f^+(x),
\end{aligned} \tag{2.74}$$

where $P_w(TP\,|\,x)$ denotes the weighted probability that the weak learner produces a true positive prediction given the sample x. Analogously for the negative predictions:

$$\begin{aligned}
\alpha_m^- &= \frac{1}{2}\ln(TN_m/FN_m) \\
&= \frac{1}{2}\ln\frac{TN_m/\sum_{n:f_m(x_n)<0} w_m^{(n)} f_m(x_n)}{FN_m/\sum_{n:f_m(x_n)<0} w_m^{(n)} f_m(x_n)} \\
&= \frac{1}{2}\ln\frac{P_w(TN\,|\,x)}{P_w(FN\,|\,x)} \\
&= \frac{1}{2}\ln\frac{P_w(y=-1\,|\,x)}{P_w(y=1\,|\,x)} = \frac{1}{2}\ln\frac{1-P_w(y=1\,|\,x)}{P_w(y=1\,|\,x)} \\
&=: f^-(x),
\end{aligned} \tag{2.75}$$

and $f^+(x) = -f^-(x)$ because of the identity

$$\ln\frac{x}{1-x} = -\ln\frac{1-x}{x}.$$

hence the weight updates result in

$$\begin{aligned}
\text{for } f_m(x)=1:\ w_{m+1}^{(n)} &= w_m^{(n)} \cdot e^{-y_n \alpha_m^+ f_m(x_n)} = w_m^{(n)} \cdot e^{-y_n f_m^+(x_n)} \\
\text{for } f_m(x)=-1:\ w_{m+1}^{(n)} &= w_m^{(n)} \cdot e^{-y_n \alpha_m^- f_m(x_n)} = w_m^{(n)} \cdot e^{-y_n f_m^-(x_n) f_m(x_n)} \\
&= w_m \cdot e^{-y_i f_m^+(x_i)},
\end{aligned} \tag{2.76}$$

which is exactly the update scheme of the RealBoost algorithm. Correspondingly, composing the single weak learners to the final classifier yields

$$\begin{aligned} F(x) &= \sum_{m:f_m(x)=1} \alpha_m^+ f_m(x) + \sum_{m:f_m(x)=-1} \alpha_m^- f_m(x) \\ &= \sum_{m:f_m(x)=1} f_m^+(x) + \sum_{m:f_m(x)=-1} -f_m^-(x) \\ &= \sum_{m:f_m(x)=1} f_m^+(x) + \sum_{m:f_m(x)=-1} f_m^+(x) \\ &= \sum_m f_m^+(x). \end{aligned}$$

□

The transformations in (2.74) and (2.75) are valid, since for a decision stump, the weighted probability estimate of producing an FP prediction given x, $P_w(FP \mid x)$, is given by the weighted proportion of false-positive samples and all samples predicted to be positive. The same holds for TP, TN and FN predictions, analogously. Equivalently, this corresponds to the weighted conditional expectations $\mathbb{E}_w(\mathbf{1}_{y=1} \mid x)$ and $\mathbb{E}_w(\mathbf{1}_{y=-1} \mid x)$, respectively.

2.5. Regularized Boosting

When discussing different loss functions in context of AdaBoost, we observed that there is the risk that the final model will be overtrained. This is due to the exponential growth of weights of noisy data that tend to be misclassified over many iterations and hence may have great influence on the training process. However, in its early days, empirical studies [6, 61, 62] suggested that AdaBoost were resistant against overfitting. Nevertheless, later studies [63, 64, 37] have shown that its immunity to overfitting is clearly a myth.

Rätsch *et al.* [37] show some interesting similarities between AdaBoost and Support-Vector Learning [52]. A first important observation in studying AdaBoost's sensitivity to noise is noting that it produces a large *hard margin* during training, i.e. boosting pursues the goal of classifying every single pattern in the training set correctly. According to maximum-margin classifiers, such as Support-Vector Machines (SVM) [51], a decision boundary is the preferable choice, that exhibits a margin as large as possible. However, because this paradigm assumes general separability of the data at

Algorithm 15 RareBoost

Start with weights $w_m^{(n)} \leftarrow {}^1/_N,\ n = 1, \ldots, N$
for $m = 1$ to M **do**
Fit the classifier $f_m(x) \in \{-1, +1\}$ using weights w_m
Compute:

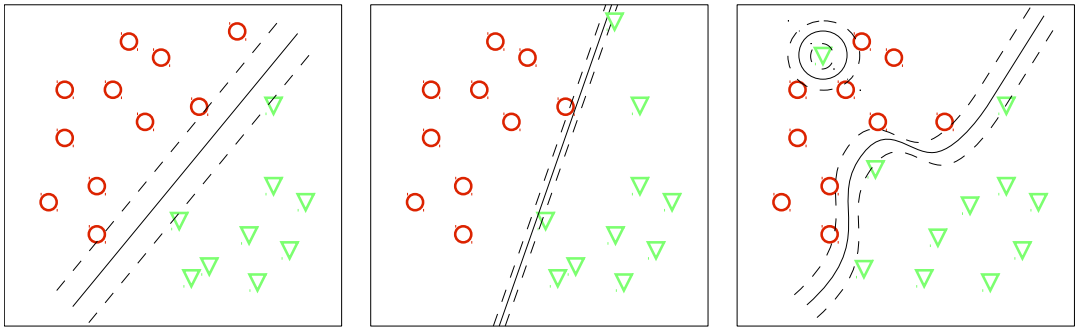
$$\begin{aligned} TP_m &= \sum_{n:f_m(x_n)=1, y_n=1} w_m^{(n)} f_m(x_n) \\ FP_m &= \sum_{n:f_m(x_n)=1, y_n=-1} w_m^{(n)} f_m(x_n) \\ \alpha_m^+ &= \frac{1}{2} \ln \frac{TP_m}{FP_m} \\ TN_m &= \sum_{n:f_m(x_n)=-1, y_n=-1} w_m^{(n)} f_m(x_n) \\ FN_m &= \sum_{n:f_m(x_n)=-1, y_n=1} w_m^{(n)} f_m(x_n) \\ \alpha_m^- &= \frac{1}{2} \ln \frac{TN_m}{FN_m} \end{aligned}$$

Update the weights:

$$\begin{aligned} \forall n : f_m(x_n) \geq 0 : \quad & w_{m+1}^{(n)} = \tfrac{1}{Z_m} w_m^{(n)} e^{-y_n \alpha_m^+ f_m(x_n)} \\ \forall n : f_m(x_n) < 0 : \quad & w_{m+1}^{(n)} = \tfrac{1}{Z_m} w_m^{(n)} e^{-y_n \alpha_m^- f_m(x_n)}, \end{aligned}$$

where Z_m is a normalization factor, such that $\sum_{n=1}^N w_{m+1}^{(n)} = 1$.
end for
Output the final Classifier $\operatorname{sgn}(F(x))$ with

$$F(x) = \sum_{m:f_m(x)\geq 0} \alpha_m^+ \cdot f_m(x) + \sum_{m:f_m(x)<0} \alpha_m^- \cdot f_m(x)$$



(a) *Noiseless Data* **(b)** *Outlier* **(c)** *Noisy data*

Figure 2.4.: *Decision boundaries of hard margin classifiers in (a) a noiseless setting, (b) presence of an outlier and (c) a noisy setting.*

hand, in the presence of noise or overlapping class distributions, hard margin training may lead to suboptimal generalization behavior.

In order to discuss the generally bad performance of hard margin classifiers in presence of unreliable data, let us analyze the toy example in Figure 2.4. In Figure 2.4(a) we sketch the case without noise. A large margin classifier (such as an SVM or AdaBoost) can successfully estimate the optimal decision boundary. If there is an outlier, however, as depicted in Figure 2.4(b), the estimate is corrupted by this single pattern, hence it has negative influence on the generalization performance of the resulting classifier. In the case of noisy feature measurements or incorrectly labeled data, as shown in Figure 2.4(c), the classifier suffers as well from hard margin training. Especially with increasing model complexity (e.g. when more and more base learners are added to the ensemble) the classifier tends to correctly classify all training samples, which results in an overfitted decision boundary and bad generalization performance [37].

We can assess the problem of hard margin classifiers in context of AdaBoost theoretically by defining the *margin* of an input-output pair $z_n = (x_n, y_n)$ as

$$\rho(x_n, \boldsymbol{\alpha}) = y_n F(x_n) = y_n \sum_{m=1}^{M} \alpha_m f_m(x_n), \tag{2.77}$$

where $\boldsymbol{\alpha}$ denotes the vector of all coefficients $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_M)^T$. This is a reasonable definition since the margin of a pattern is commonly understood as its "distance" to the decision boundary [51]. Additionally, we can define the margin of a whole classifier $F(x)$ as the minimal margin of all training patterns, i.e.

$$\varrho(\boldsymbol{\alpha}) = \min_n \rho(z_n, \boldsymbol{\alpha}).$$

Using the definition in (2.77), we can write the exponential loss function in terms of the margin:

$$L\left(y, \rho(x, \boldsymbol{\alpha})\right) = \sum_{n=1}^{N} e^{-\rho(x_n, \boldsymbol{\alpha})}, \tag{2.78}$$

which we already minimized analytically in previous sections. The important observation is that L essentially defines a loss function over margin distributions, which depends on the α coefficients, i.e. the larger the margins, the smaller the loss will be. So, in order to decrease L maximally, α and the hypothesis should be selected in such a way, that the margins increase most strongly. Rätsch *et al.* [37] interpret this as an analogy to support vector learning, and they show that the smallest margin of patterns of each class asymptotically converge to the same value. In the style of SVMs, they call these patterns with smallest margin *support patterns*.

AdaBoost$_{Reg}$

In order to overcome the problem of overfitting, the SVM model is enhanced by *slack variables* that allow a certain misclassification error during the training phase in an area close to the margin, which results in a smoother decision boundary and counters the problem of such "insular" classification regions as shown in Figure 2.4(c). The dose of the tolerated error is controlled by a regularization parameter C. An analogous extension to AdaBoost exists which we want to introduce in the following.

As can be seen from (2.78), the loss function is minimized when the margin ρ is maximized, where for all pattern margins

$$\rho(x_n, \boldsymbol{\alpha}) \leq \varrho(\boldsymbol{\alpha})$$

holds. If $\varrho > 0$, then all $-$ also the possibly mislabeled or noisy $-$ patterns are predicted correctly, which indicates overfitting in presence of noise. Therefore, a modification to AdaBoost should allow these unreliable patterns to have a margin smaller than 0. If we knew them beforehand, we could just remove them from the data set, but in practice, identifying these patterns is difficult. Suppose we have a measure $\xi(x_n)$ that denotes the "mistrust" we have in a particular sample x_n. This could be a probability that x_n is an outlier or mislabeled, for instance. Then we can relax the above inequality by

$$\rho(x_n, \boldsymbol{\alpha}) \leq \varrho(\boldsymbol{\alpha}) - C\xi(x_n),$$

such that a highly mistrusted sample may have a margin smaller than ϱ, where C is an a-priori chosen constant. Additionally, we can define the *soft margin* of a pattern as

$$\tilde{\rho}(x_n, \boldsymbol{\alpha}) = \rho(x_n, \boldsymbol{\alpha}) + C\xi(x_n), \qquad \text{where } \tilde{\rho}(x_n, \boldsymbol{\alpha}) \leq \varrho(\boldsymbol{\alpha}). \tag{2.79}$$

Now the boosting procedure simply has to be modified to maximize the soft margin instead of the hard margin. However, there remains the problem of how to estimate the mistrust function $\xi(x_n)$. Here, good heuristics are in demand. Rätsch *et al.* [37] propose the choice of ξ to be based on the influence of a pattern on the combined hypotheses f_m

$$\xi(x_n) = \sum_{m=1}^{M} \alpha_m w_m^{(n)}, \tag{2.80}$$

which is a weighted sum of a pattern's weights over the boosting rounds. Following these heuristics, a sample that is frequently misclassified during training (which is seen as an indication for noise), will have a large average weight and thus is supposed to be untrustworthy.

The new loss function in terms of the soft margin becomes [37]

$$\begin{aligned} L_{Reg}(y, \boldsymbol{\alpha}^m) &= \sum_{n=1}^{N} \exp\left(-\frac{1}{2}\tilde{\rho}(x_n, \boldsymbol{\alpha}^m)\right) \\ &= \sum_{n=1}^{N} \exp\left(-\frac{1}{2}\left[\rho(x_n, \boldsymbol{\alpha}^m) + C|\boldsymbol{\alpha}^m|\mu_m(x_n)^p\right]\right), \end{aligned} \tag{2.81}$$

where $\boldsymbol{\alpha}^m$ denotes the vector of α coefficients up to the current iteration m, $\boldsymbol{\alpha}^m = (\alpha_1, \ldots, \alpha_m)^T$ and p is an a-priori chosen constant (e.g. $p \in \{1, 2\}$). Unfortunately, (2.81) does not have an analytical solution. Rätsch *et al.* propose to employ a line search procedure [72] for obtaining the optimal α in each iteration.

A complete algorithmic description of the AdaBoost$_{Reg}$ procedure is given in Algorithm 16.

WeightBoost

So far, we supposed the ensemble obtained by the AdaBoost algorithm (and its derivatives) to take the form of Eq. (2.16), i.e. an additive model. In order to get to a model of this kind we greedily learned a hypothesis $f_m(x)$ in each iteration and assigned it a mixing coefficient α_m. Except for the LPBoost and QPBoost variations [37], none

Algorithm 16 AdaBoost$_{Reg}$

Start with weights $w_0^{(n)} \leftarrow 1/N,\ n = 1, \ldots, N$
for $m = 1$ to M **do**
Fit the classifier $f_m(x) \in \{-1, +1\}$ using weights w_m
Compute

$$\alpha_m = \arg\min_{\alpha_m} \sum_{n=1}^{N} \exp\left(-\frac{1}{2}\left[\rho(z_n, \boldsymbol{\alpha}^m) + C|\boldsymbol{\alpha}^m|\mu_m(z_n)^p\right]\right), \tag{2.82}$$

$$\text{with } \rho(z_n, \boldsymbol{\alpha}^m) = y_n \sum_{t=1}^{m} \alpha_t f_t(x_n),\ \boldsymbol{\alpha}^m = (\alpha_1, \ldots, \alpha_m)$$

Update the weights:

$$w_{m+1}^{(n)} = \frac{1}{Z_m} \exp\left(-\frac{1}{2}\left[\rho(z_n, \boldsymbol{\alpha}^m) + C|\boldsymbol{\alpha}^m|\mu_m(z_n)^p\right]\right)$$

where Z_m is a normalization factor such that $\sum_{n=1}^{N} w_{m+1}^{(n)} = 1$
end for
Output the final Classifier $\text{sgn}(F(x))$ with

$$F(x) = \sum_{m=1}^{M} \alpha_m f_m(x)$$

of the boosting algorithms touches any of the parameters that have already been determined in a previous iteration, where α_m, $(1 \leq m \leq M)$, is a constant, such that the final model is a linear combination of weak hypotheses:

$$F(x) = \sum_{m=1}^{M} \alpha_m f_m(x)$$

This approach is an implementation of the *classifier fusion* paradigm we introduced in Chapter 1, i.e., for classifying a new data point, each weak hypothesis gives its vote, and all of the individual decisions are combined by a weighted majority vote. An alternative paradigm in combining classifier outputs is to select that one from which we expect the most promising response, based on the current sample at hand. More formally, the weighting coefficients become functions that depend on the input themselves:

$$F(x) = \sum_{m=1}^{M} \gamma_m(x) f_m(x)$$

A model of this form is also known as a *mixture-of-experts* and can be employed in order to alleviate the overfitting problem. The principal idea of achieving this is as follows: As pointed out earlier, the cumulative misclassification by multiple base classifiers and the related exponential growth of weights is a reason why noisy data may distort the training process of boosting. At the same time, as noise is still supposed to occur only desultorily over the whole input space, mistrusted patterns are expected to be misclassified with high confidence, i.e. their margin is expected to be significantly below 0 (because they are supposed to be surrounded by a vast amount of trustworthy patterns). Now, if the mixing coefficient $\gamma_m(x)$ is chosen to be inverse to the previously accumulated weights, we should be able to filter out noisy data from the training process, since, as their weights get larger and larger, they forfeit their influence on future boosting iterations.

In the *WeightBoost* algorithm proposed by Jin *et al.* [38], the mixing function $\gamma_m(x)$ is defined as

$$\gamma_m(x) = \alpha_m e^{-|\beta F_{m-1}(x)|}, \tag{2.83}$$

where α_m denotes a mixing coefficient as in original AdaBoost, β is a regularization parameter and $F_{m-1}(x)$ denotes the ensemble output up to the current iteration m.

By minimizing the upper bound on the training error in the style of Schapire and Singer [31], one obtains the choice for α_m

$$\begin{aligned}\alpha_m &= \frac{1}{2}\ln\frac{\sum_{n=1}^{N} e^{-F_{m-1}(x_n)y_n} e^{-|\beta F_{m-1}(x_n)|}\mathbf{1}_{f_m(x_n)=y_n}}{\sum_{n=1}^{N} e^{-F_{m-1}(x_n)y_n} e^{-|\beta F_{m-1}(x_n)|}\mathbf{1}_{f_m(x_n)\neq y_n}} \\ &= \frac{1}{2}\ln\frac{\sum_{n=1}^{N} w_m^{(n)}\mathbf{1}_{f_m(x_n)=y_n}}{\sum_{n=1}^{N} w_m^{(n)}\mathbf{1}_{f_m(x_n)\neq y_n}} \\ &= \frac{1}{2}\ln\frac{1-\epsilon_m}{\epsilon_m},\end{aligned}$$

where we have redefined the weights as

$$w_m^{(n)} = \frac{1}{Z_m} e^{-F_{m-1}(x_n)y_n} e^{-|\beta F_{m-1}(x_n)|}$$

and ϵ_m refers to the weighted misclassification error produced by the m-th weak learner. The complete boosting procedure is listed in Algorithm 17.

Algorithm 17 WeightBoost

Start with weights $w_1^{(n)} \leftarrow 1/N,\ n = 1,\ldots,N$
for $m = 1$ to M **do**
Fit the classifier $f_m(x) \in \{-1,+1\}$ using weights w_m
Compute

$$\alpha_m = \frac{1}{2}\ln\frac{1-\epsilon_m}{\epsilon_m}, \qquad \text{where } \epsilon_m = \sum_{n=1}^{N}\mathbf{1}_{y_n\neq f_m(x)}$$

Update the weights: $w_{m+1}^{(n)} = \frac{1}{Z_m} e^{-y_n F_m(x_n) - |\beta F_m(x_n)|}$,
where Z_m is a normalization factor, such that $\sum_{n=1}^{N} w_{m+1}^{(n)} = 1$
end for
Output the final Classifier $\operatorname{sgn}(F(x))$ with

$$F(x) = \sum_{m=1}^{M} \alpha_m e^{-|\beta F_{m-1}(x)|} f_m(x)$$

2.6. Summary

In the previous sections we introduced some of the most outstanding developments in the area of boosting that have been reported from its early days up to present.

We introduced the original (discrete) AdaBoost algorithm which relies on base classifiers with binary output, as well as its variants RealBoost, GentleBoost and LogitBoost that use weighted probability estimates instead of discrete class assignments. Since the original boosting algorithms do not differentiate between class-specific misclassification errors, we have discussed several cost-sensitive extensions which have been designed in order to cope with the problem of imbalanced classes and uneven misclassification costs.

We gave theoretical explanations of phenomena that have only been empirically observed by other authors [36, 21, 34]:

- On page 51, we illustrated why the AdaC1 algorithm is instable in particular situations since introducing cost items inside the exponent of the weight updates does not result in an effective bias towards the respective class. Exponential cost factors in the weight updates rather cause the weights to oscillate since stronger penalties of misclassifications are compensated by stronger rewards of correct predictions.
- At population level, we have derived Eq. (2.69) on page 58 which relates exponential cost settings in the expected exponential loss to the weighted conditional probability estimate that the base learner is supposed to fit. Here, we found that exponential cost setups regarding a particular class causes the base classifier to be biased towards the other class during training, but the biasing effect vanishes due to the ensemble update $f_m(x)$ in Eq. (2.68). Hence we have shown that this approach, which has been first proposed by Masnadi-Shirazi and Vasconcelos [39], is not recommended.
- Another cost-sensitive approach, AdaCost, strives to compensate for the drawbacks of the AdaC1 algorithm by introducing a cost adjustment function in the exponent of the weight updates, which prevents the weights from oscillating. On page 46 we gave a theoretical explanation of the still poor performance improvement by AdaCost, which has been empirically observed by several authors [21, 34]. Due to the fact that there is no cost setting for which AdaCost reduces to original AdaBoost and costs are bounded by the interval $[0, 1]$, the semantics of costs in this algorithm are counterintuitive, which makes it less practical.
- For the RareBoost algorithm, which does not employ explicit cost parameters, we have shown that it is a special case of the RealBoost algorithm. In Proposition 3, we have even proven that these two methods are equivalent in the case

where decision stumps are used as base learners. In Lemma 1 and Proposition 2, we have also disproved the presumption given in [21] that RareBoost were instable in particular situations.

- Based on our theoretical findings in this chapter we conclude that for countering the imbalance problem employing cost-sensitive boosting variants that use multiplicative cost factors in their weight updates are more favorable, such as the AdaC2 algorithm and its corresponding probabilistic methods CSRA, CSGA and CSLB. We confirm this theoretical result in the next chapter by an empirical study.

Since the cost-sensitive variants and the original AdaBoost procedures are prone to overfitting in noisy settings and in presence of overlapping class distributions, we also considered two regularized versions, namely AdaBoost$_{Reg}$ and WeightBoost. Both techniques pursue different strategies of regularization. AdaBoost$_{Reg}$ adopts the concept of soft margins similar to Support-Vector-Machines by assigning each training pattern a "mistrust" that controls its influence on the subsequent training iterations. By contrast, WeightBoost implements a mixture-of-experts system that mediates between influence of different ensemble members depending on the input pattern. In the next chapter we show that these two approaches lead to different behavior in presence of rare classes.

We conclude our survey of state-of-the-art boosting techniques with a concise juxtaposition of all algorithms considered in this chapter, which can be found in Table 2.1. In the next chapter, we continue with our discussion of boosting and empirically study their performance and ability to cope with rare classes and concurrent goals of classification.

Approach	Algorithm	Cost-Sensitivity	Regularization
Data Level	(Discrete) AdaBoost	−	−
	CSB0, CSB1, CSB2	+	−
	AdaC1, AdaC2, AdaC3	+	−
	AdaCost	+	−
	RareBoost	−	−
	AdaBoost_{Reg}	−	+
	WeightBoost	−	+
Population Level	RealBoost	−	−
	GentleBoost	−	−
	LogitBoost	−	−
	CSRA	+	−
	CSGA	+	−
	CSLB	+	−

Table 2.1.: *Overview of boosting algorithms considered in Chapter 2.*

Chapter 3

Empirical Evaluation

In this chapter, we study the performance of boosting algorithms and their derivations introduced in previous chapters. We empirically compare and evaluate the algorithms concerning their ability to cope with the imbalance problem in terms of pure numerical imbalance as well as competitive classification goals, their generalization performance and means of regularization, and their robustness against noise and overlapping class distributions. In particular, we address the following issues:

1. *Impact of imbalance.* In previous chapters, we referred to imbalanced class distributions and competitive classification goals as a problem making classification harder than it is when dealing with even classes. In this chapter, we show how imbalance affects the performance of classical boosting algorithms which have not been endowed with additional means for rebalancing a-priori class distributions.

2. *Impact of cost-sensitivity.* As we have shown in the previous chapters, the only algorithmic approach in countering the imbalance problem so far is introducing cost items into the weight update rule of AdaBoost. We examine how cost-sensitive methods are able to alleviate imbalance and can be used to bias a classifier in order to achieve a particular goal of classification, when class preferences are uneven.

3. *Regularization and overfitting.* In its early days, AdaBoost gained a lot of attention because it conveyed being immune against overfitting. Later it was shown by Rätsch [37] and others, that boosting shares many properties with support-vector machines, which indeed tend to overfit unless being equipped with some means of regularization. We study the regularization behavior in presence of noise and overlapping class distributions, in particular of cost-sensitive variants. We also examine how regularized methods behave in presence of the imbalance problem.

Up to this thesis, to the best of our knowledge, we do not know any work providing an extensive analysis and contrasting juxtaposition of a wide range of boosting algorithms, including cost-sensitive methods as well as regularized variants. Although there has been done a lot of work on the topic of cost-sensitive extensions of AdaBoost there is no study on their behavior in presence of noise and their tendency to overfit. Likewise, regularized methods have not been regarded in presence of rare classes so far and imbalance has only been considered in terms of numerically uneven a-priori distributions, disregarding the ubiquitous case of aiming at particular class-specific error rates.

Before discussing the conducted experiments and results we first provide a description of the data we use for our empirical studies.

3.1. Evaluation Setup and Data

Most authors of previous work evaluate their algorithms on synthetic toy data or academic benchmark data sets. However, the ultimate goal of machine learning techniques is to turn up in practical real-world applications. However, many of the boosting algorithms reported in literature have only sparsely been empirically evaluated on real-world data. As we already mentioned in Chapter 1, in practice, imbalanced classes are ubiquitous but, since imbalance often occurs as a problem-intrinsic property, it can hardly be reflected adequately by synthetically generated data. Hence, we put emphasis on evaluating the algorithms considered in this work on real-world data collected in different industrial environments and applications at Siemens AG, as well as on two publicly available benchmark data sets.

Network Intrusion Detection

Network Intrusion Detection Systems (IDS) are systems for automatically detecting computer attacks. For instance, computer attacks may aim at eaves-dropping communication, tampering with files of compromised hosts or misuse of hardware resources [67]. The data we are using have been ascertained by Rieck [67] and consist of two major sets containing 10 days of consecutive network traffic of HTTP (Hypertext Transfer Protocol) (data set 1) and FTP (File Transfer Protocol) (data set 2) servers, which have been enhanced by adding data of different network attacks. Each instance in the data set represents a network connection by means of seven statistical features:

1. Length of payload: `real`
2. Byte entropy of payload: `real`
3. Minimum byte value in payload: `real`
4. Maximum byte value in payload: `real`
5. Number of distinct bytes: `real`
6. Number of non-printable characters: `real`
7. Number of punctuation characters: `real`

The data contain more than 50,000 patterns and are highly imbalanced from a numerical point of view. 98.82% of the data represent "normal" network connections ($y = -1$), whereas network attacks are represented by only 1.18%. The classification goal is to identify as much network attacks as possible, whereas false alarms are sternly to be avoided, since, in the view of the drastic numerical disproportion of normal connections ($y = -1$) and attacks ($y = +1$), even a false-positive rate of only 1%, for instance, would cause an unacceptably high number of connections that are potentially refused in a system in operation. Both the HTTP and FTP data set consist of 20 data subsets, each of which contains about 2530 (including 30 attacks) patterns. For each subset there is an independent test set provided for evaluation. The results always are averaged over all 20 subsets. As base classifiers decision stumps are used.

Fire Detection

In next generation industrial fire detection systems, chemical gas sensors are deployed that are changing their electrical properties when reacting with gases that are formed in case of fire. Currently, there are three chips of different materials under investigation, i.e. platinum, titan nitride and phtalocyanine. Hence, each data instance is represented by a three-dimensional vector holding the gradient of induced voltage measured by each sensor:

1. Pt (platinum): `real`
2. TiN (titan nitride): `real`
3. PH (phtalocyanine): `real`

The data are smoothed by moving average filtering (20 samples window size) and resampled afterwards. Although the samples originate from a time series, they are assumed i.i.d. for classification. In the selection of data sets considered in this work, this is the only one representing a three-class problem, in particular

1. clean air: $y = 0$

2. open fire: $y = 1$
3. smoldering fire: $y = 2$,

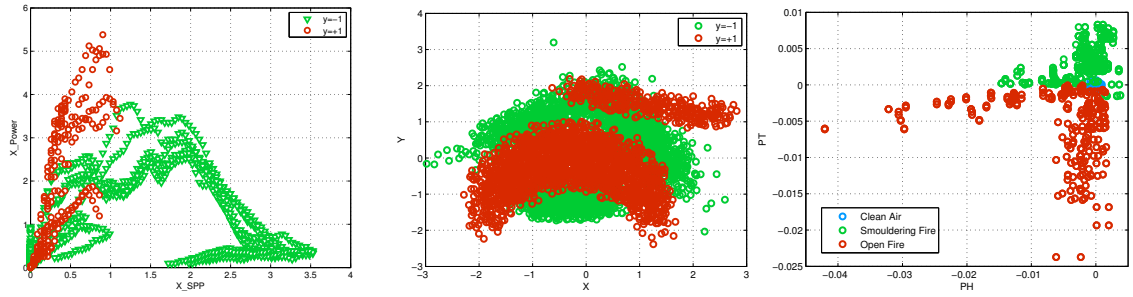
which allow further distinction between different types of fire. Since we do not consider multiclass problems in this work, and the primary goal of classification is discriminating between *fire* and *no-fire*, we subsume the *open* and *smoldering* fire cases in one single class *fire*. Fire patterns ($y = +1$) are represented by 12.5% of the data, 87.5% are no-fire cases. The set consists of about 1,600 samples. Since no test sets are provided, we use 5-fold cross validation, which we run 10 times and average the results using stumps as base learners.

Airbag Deployment & Crash Detection

In safety-related applications, such as automotive crash detection systems, where a control system has to decide whether or not to trigger the airbag, it is particularly important not to produce false-positive predictions [68]. However, not deploying it in real crash situations may have disastrous consequences as well. Additionally, besides this kind of imbalance in terms of uneven misclassification costs, eliciting enough crash data is elaborate and costly, hence much more "non-crash" data (which are given by negative examples $y = -1$) are available than "crash" data (positive examples, $y = +1$), such that the data exhibit strong numerical imbalance, too. The original data set provided by Nusser *et al.* [68] consists of 30-dimensional vectors per pattern, but we use a 2-dimensional projection, where one variable can be interpreted as a temporal quantity, the other one as the corresponding measurements of the negative acceleration of the vehicle. There are two data subsets available, each comprising 2,500 patterns, 12.0% of which represent "fire" cases. Since no test sets are available, we use 5-fold cross validation, averaged over 10 runs. The base learning algorithm used with this data is CART.

Banana

The Banana data are an artificial data set by Rätsch [37] synthetically generated for benchmark purposes. The 2-dimensional data are generated by multiple gaussian distributions that form four convoluted "bananas" with partially overlapping areas. Although these data are not highly imbalanced they are suitable for comparing different algorithms regarding regularization and overfitting aspects. 100 subsets are provided, each consisting of one training set and one corresponding test set. For computational reasons, however, we pick only 10 train-test pairs out of the 100.



(a) Crash Detection *(b)* Banana *(c)* Fire Detection

Figure 3.1.: *Visualization of two dimensional data sets (a) airbag control, (b) banana and a 2-dimensional projection of the (c) fire detection data.*

Wisconsin Breast Cancer

Wisconsin Breast Cancer data [69] obtained from the *UCI Machine Learning Repository* [70] is a popular data set often called on in literature for benchmarking purposes. The classification task is to discern malignant (positive class, $y = +1$) from benign (negative class, $y = -1$) cases. The 9 numerical variables represent morphologic and biochemical features extracted from a digitized image of a fine needle aspirate (FNA) of a breast mass, one per instance. The features are given by

1. Clump Thickness: `real`
2. Uniformity of Cell Size: `real`
3. Uniformity of Cell Shape: `real`
4. Marginal Adhesion: `real`
5. Single Epithelial Cell Size: `real`
6. Bare Nuclei: `real`
7. Bland Chromatin: `real`
8. Normal Nucleoli: `real`
9. Mitoses: `real`

The data set contains 699 patterns, 241 of which (35.0%) represent the malignant ($y = +1$) case. There are 16 instances with incomplete feature information, these are removed from the data, since the base learners that are used have no means for dealing with unknown or nominal features. Figure 3.1 shows a visualization of the two dimensional classification problems considered in this work. The most important properties of the data set are summarized in Table 3.1.

Data Set	Variables	Imbalance	Important Class	Base Learner
IDS Data	7	1.1% vs. 98.82%	y=−1 (no attack)	stump
Fire Detection	2	12.5% vs. 87.5%	y=−1 (clean air)	stump
Airbag Control	2	12.0% vs. 88.0%	y=−1 (no crash)	CART
Breast Cancer	9	35.0% vs. 65.0%	y=+1 (malignant)	stump
Banana 100	2	42.6% vs. 57.4%	N/A	CART

Table 3.1.: *Summary of all data sets considered in this work and the base classifiers that are used in each case.*

Performance Measure

As pointed out in Chapter 1, there are several metrics for measuring the performance of a classification system and not all of them are suitable in case of imbalanced classes. As most commonly employed measures in literature we use precision, recall and F1-score for determining the classification performance of the algorithms. In cases where it is appropriate, we also bring in the misclassification error with respect to the training and test set performance.

Some of the algorithms considered in this work are driven by some parameter, such as cost factors or regularization constants. Unfortunate choices of these variables result in poor performance and hence an inadequate validation. In most empirical studies of boosting, equal choices of parameter values are applied to different algorithms [21, 36, 28, 39]. Since such model parameters have to be chosen individually for each problem and optimal choices may vary from case to case, we think that this way of comparing different methods does not necessarily lead to reasonable results. In order to assess the performance of each algorithm as fair-minded as possible, we determined a range of 10 parameter values for each algorithm and problem, in which it performs best. The parameter ranges have been manually determined in advance by 5-fold cross validation on the training set. In doing so, we give each approach the chance to get on its best behavior.

3.2. Experiments

In this section we present the results of the experiments that have been conducted in order to assess the algorithms' performances in the scenarios described above. In each case we only pick some of the most outstanding results. For a complete list of evaluation statistics we refer to Appendix B.

For the experiments, we use two kinds of base learners: decision stumps and Classification and Regression Trees (CART) [47]. Both are identical with regard to their learning procedures, yet different in terms of bias and variance. A decision stump can be seen as a special case of a CART that has only one single terminal node, therefore is characterized by high bias. In contrast, as CART learning can build a decision tree down to each single training pattern, it is highly sensitive to changes in the data. Since the AdaBoost algorithm is supposed to be *adaptive* to any kind of classifier, even those that are just slightly better than random guessing, we think that these two extremes are a good choice for getting a big picture of AdaBoost's performance. In each of the data sets all algorithms are run with the same base learner and the same number of iterations in order to ensure comparability.

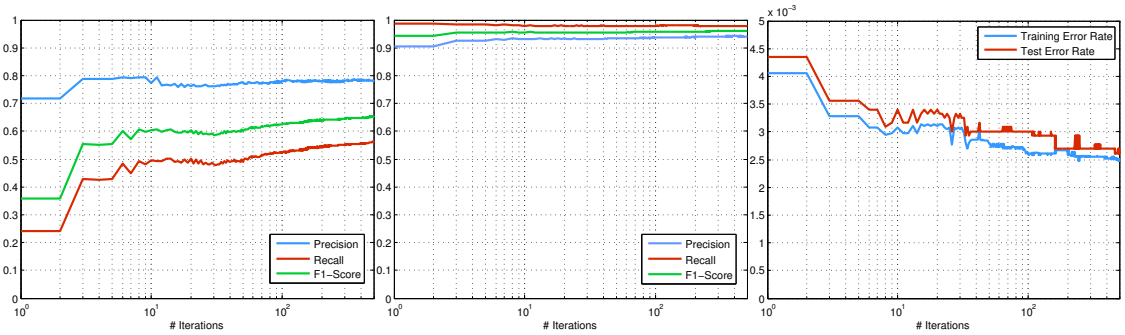
3.3. Results

Impact of Imbalance

In this section we demonstrate how numerical imbalance affects the classification performance of algorithms that do not have additional mechanisms for rebalancing the a-priori distributions. For this purpose we run the cost-insensitive boosting methods *AdaBoost*, *RealBoost*, *GentleBoost* and *LogitBoost* on the data sets that exhibit strong numerical imbalance, i.e. IDS, airbag and cancer data.

Cost-Insensitive Methods

We exemplarily show the progression of precision, recall and F1-score of the positive and the negative class in Figure 3.2, as well as the training and test error rates of the (Discrete) AdaBoost algorithm applied to the airbag data set using CART over 500 iterations. As can be clearly seen, while all 3 measures regarding the negative class



(a) $y = +1$ **(b)** $y = -1$ **(c)** *Error Rates*

Figure 3.2.: *Precision, recall and F1-score with respect to the (a) positive and (b) negative class. (c) the corresponding training and test error rates, achieved by Discrete AdaBoost on the crash data over 500 iterations.*

(Figure 3.2(b)) lie significantly above 0.9 from the first iteration on, the F1-score of the positive class ranges from below 0.4 to 0.6.

As can be seen in Table 3.2, the cost-insensitive boosting variants perform almost equally, though a slight tendency can be detected, that the probabilistic variants, especially the GentleBoost and RealBoost variants outperform Discrete AdaBoost, which confirms the empirical findings of Friedman *et al.* [11]. Additionally, as a result of the bias towards the prevalent class, which is much more frequently predicted than the minority class, mainly recall with respect to the rare class suffers from imbalance. In contrast, precision is relatively high, since only few instances are classified as the minority class and as a consequence, there are only few FP cases produced by the classifier.

Regularized Methods

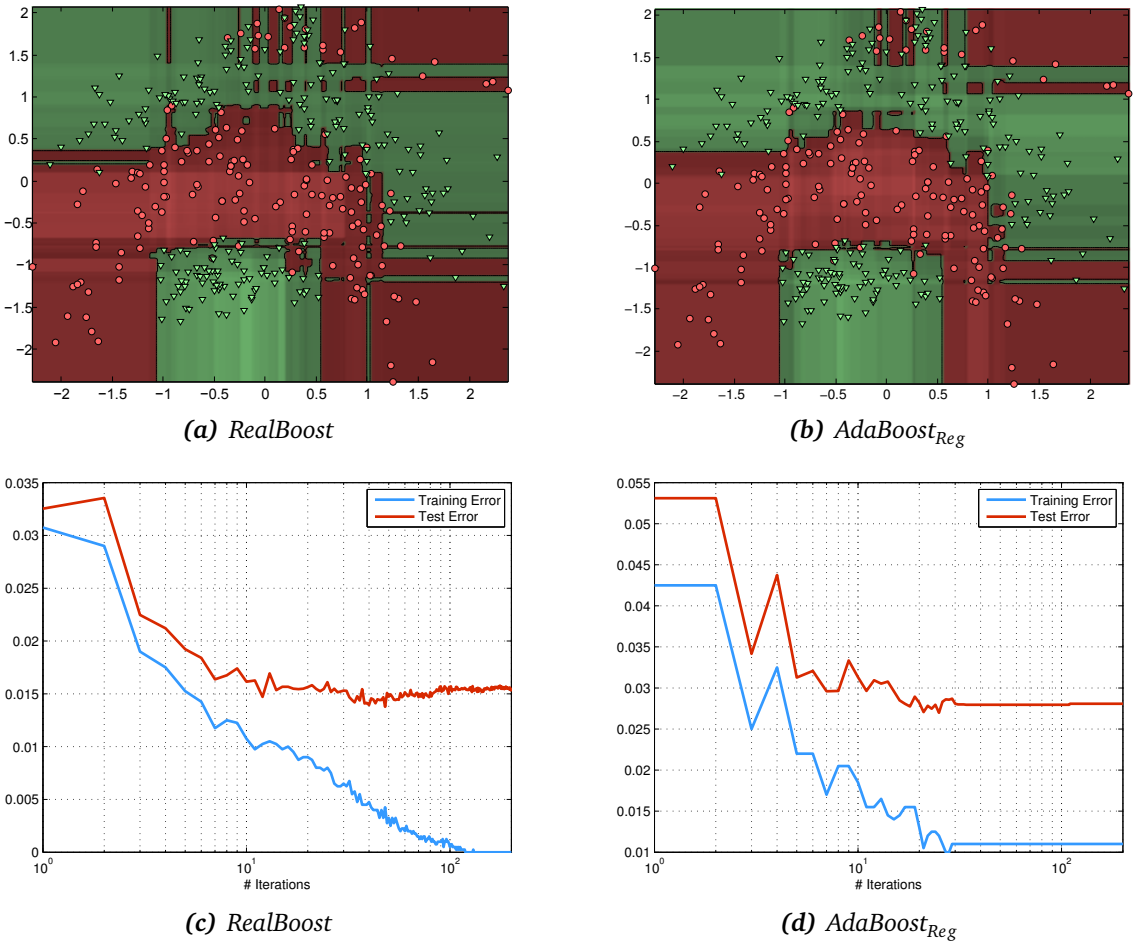
For analyzing how imbalance affects the behavior of the regularized algorithms we concentrate on two different data sets which both represent two-dimensional problems, namely the banana benchmark as well as the airbag data. We choose these two data sets since two dimensional data are easy to visualize and therefore enables convenient evaluation of the algorithms' performance. Additionally, both data sets contain overlapping class distributions, which is a property only necessitating regularization.

We first analyze the behavior of the classical boosting algorithms and show that AdaBoost, against the common predominant assumption, indeed tends to overfit in pres-

Algorithm		AdaBoost			RealBoost		
Data Set		P	R	F	P	R	F
Banana	y=+1	0.818	0.835	**0.826**	0.814	0.827	0.820
	y=-1	0.865	0.849	**0.857**	0.859	0.846	0.853
IDS	y=+1	0.970	0.860	0.907	0.973	0.869	**0.910**
	y=-1	0.998	1.000	**1.000**	0.998	1.000	0.999
Cancer	y=+1	0.942	0.926	0.933	0.942	0.930	0.936
	y=-1	0.961	0.968	0.964	0.963	0.969	0.966
Fire	y=+1	0.994	0.999	0.997	0.997	0.999	**0.998**
	y=-1	0.998	0.966	0.981	0.997	0.984	**0.990**
Airbag	y=+1	0.700	0.529	0.600	0.711	0.541	0.612
	y=-1	0.937	0.969	0.953	0.939	0.969	**0.954**

Algorithm		GentleBoost			LogitBoost		
Data Set		P	R	F	P	R	F
Banana	y=+1	0.815	0.833	0.823	0.813	0.838	0.825
	y=-1	0.863	0.847	0.855	0.866	0.844	0.855
IDS	y=+1	0.974	0.854	0.905	0.936	0.847	0.880
	y=-1	0.998	1.000	0.999	0.998	0.999	0.999
Cancer	y=+1	0.942	0.939	**0.940**	0.948	0.930	0.938
	y=-1	0.968	0.968	**0.968**	0.963	0.972	0.967
Fire	y=+1	0.995	0.999	0.997	0.995	0.999	**0.998**
	y=-1	0.995	0.972	0.982	0.997	0.976	0.986
Airbag	y=+1	0.717	0.541	**0.614**	0.713	0.539	0.612
	y=-1	0.939	0.970	**0.954**	0.938	0.970	**0.954**

Table 3.2.: *Precision (P), Recall (R) and F1-score (F) for the positive and negative class of cost-insensitive boosting algorithms applied to the data sets. Highest F1-scores are bolded. The values are rounded to 3 decimals.*



(a) *RealBoost* *(b)* *AdaBoost*$_{Reg}$

(c) *RealBoost* *(d)* *AdaBoost*$_{Reg}$

Figure 3.3.: *Decision boundaries and progress of error of the RealBoost and AdaBoost*$_{Reg}$ *algorithms. The brightness of the colored areas indicates the confidence.*

ence of overlapping class distributions, and hence we confirm our expectation driven by the theoretical analyses of the previous chapter.

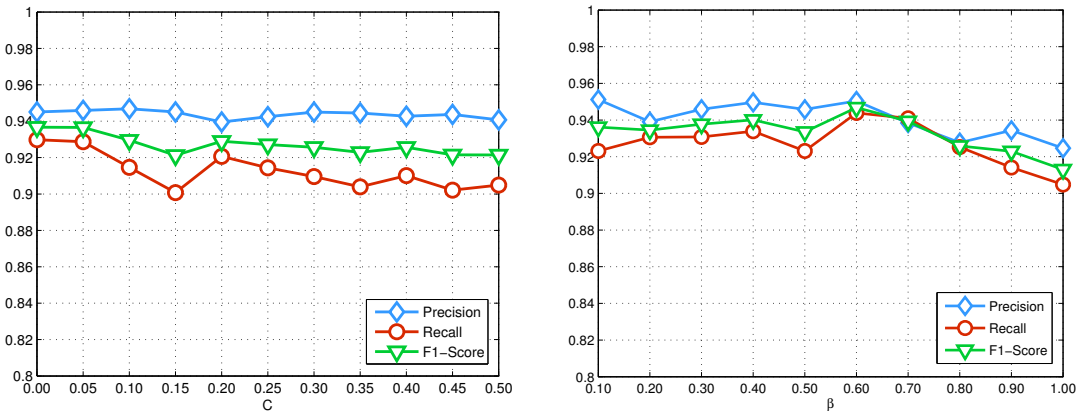
Figure 3.3 shows the decision boundaries learned by the RealBoost and the AdaBoost$_{Reg}$ algorithms applied to the Banana data set. The brightness of the colored areas indicates the confidence, i.e. the absolute value of the ensemble output $|F(x)|$. As several "insular" regions can be detected in the output of RealBoost, where not even one training pattern of the particular class is located, one might say that the classifier shown in Figure 3.3(a) suffers from overfitting. Studying the progress of training and generalization error in Figure 3.3(c), it can be seen that the training error reaches zero after 100 iterations, whilst the generalization error increases from iteration 50 on. Comparing to the classifier learned by the AdaBoost$_{Reg}$ procedure, shown in Figure 3.3(b), we detect that its boundary is obviously smoother and not as frayed as

Algorithm		AdaBoost$_{Reg}$			WeightBoost		
Data Set		P	R	F	P	R	F
Banana	y=+1	0.833	0.833	**0.832**	0.820	0.835	0.827
IDS	y=+1	0.998	0.703	0.812	0.964	0.855	**0.900**
Cancer	y=+1	0.945	0.930	0.937	0.950	0.944	**0.946**
Fire	y=+1	0.991	0.993	0.992	0.995	1.000	**0.997**
Airbag	y=+1	0.716	0.536	0.611	0.735	0.569	**0.639**

Table 3.3.: *Precision (P), Recall (R) and F1-score (F) for the positive class of regularized boosting algorithms applied to the data sets. Highest F1-scores are bolded. The values are rounded to 3 decimals. In each case the result obtained by the best parameter setting is shown.*

in the RealBoost case. In terms of misclassification, the training error stagnates at a value significantly above 0 after 30 iterations and so does the generalization performance. However, though higher misclassification error, the regularized method outperforms the unregularized one clearly, as can be seen by inspecting the F1-scores in Table 3.2 and 3.3.

Let us now examine the differences between the AdaBoost$_{Reg}$ and the WeightBoost procedures. Table 3.3 summarizes the results of the two regularized boosting methods. As can be seen, AdaBoost$_{Reg}$ performs quite similar to Discrete AdaBoost or might be slightly better in some cases. However, as the regularization parameters have been chosen to be as benignly as possible, it turned out that AdaBoost$_{Reg}$ shows its best performance at parameter choices very close to 0 in strongly imbalanced settings, i.e. when the regularization terms in its weight update scheme vanish and it reduces to the original update. We illustrate this effect in Figure 3.4(a), where we show AdaBoost$_{Reg}$'s performance at different operating points (i.e. different assignments of the regularization factor C). The figure shows decreasing performance in increasing choices for C. Indeed, during our experiments it turned out that the algorithm dramatically suffers from choices of C that are even larger. We explain this behavior with the equal treatment of both the minority and the majority class. AdaBoost$_{Reg}$ is designed to allow for misclassifications close to the decision boundary in a certain degree in order to avoid overfitting. The intensity of regularization of each pattern is determined by means of an estimate of "mistrust", given by its sum of weights, averaged over previous iterations. In presence of imbalance, however, due to the bias towards the prevalent class, minority patterns are expected to be misclassified frequently, leading to exponentially increasing weights. Hence they are assigned huge mistrust and, as a consequence, loose impact on the training process. One might say, the minority class is "regularized away".



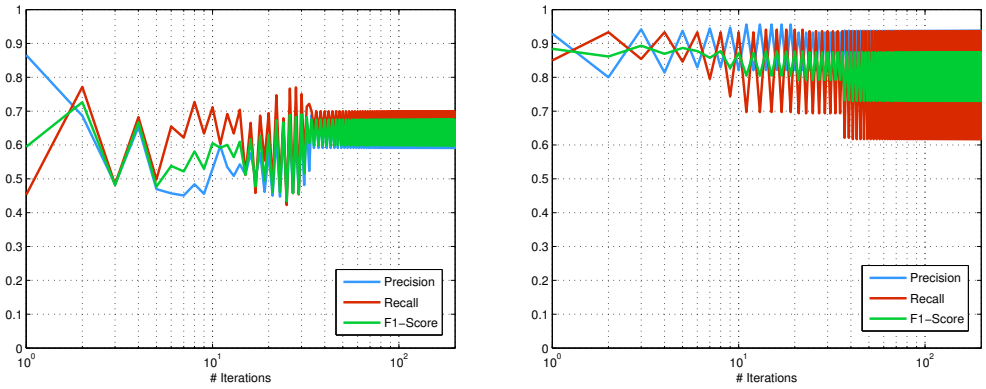
(a) $AdaBoost_{Reg}$ *(b)* *WeightBoost*

Figure 3.4.: *Precision, Recall and F1-score of the original* $AdaBoost_{Reg}$ *and WeightBoost with at different operating points applied to the cancer data. Different choices of parameter values of the respective algorithms are spread over the x-axis.*

We know from Table 3.3 that WeightBoost significantly outperforms $AdaBoost_{Reg}$, applied to almost all data sets, except the banana data. This is remarkable since it achieves considerably higher F1-scores than the classical variants in three out of five data sets as well. We make an attempt to explain the dominance of WeightBoost over $AdaBoost_{Reg}$ as follows: Based on the confidence a training pattern is predicted with up to a particular iteration, it looses influence on the training process due to the margin, which is further increased, so its weight is scaled down, as it is in the $AdaBoost_{Reg}$ update scheme. However, in the latter case, weights of such a pattern may re-increase due to misclassification by subsequent base learners. In WeightBoost, indeed, this does not happen since the influence of subsequent base learners is scaled down as well by the γ-function, such that future misclassifications have no more impact. This can be thought of as a kind of "permanent smooth removal" of samples from the data that are easy to classify. In an imbalanced setting, this procedure may lead to removal of large portions of the prevalent class, which is likely to be predicted with large confidence, such that the class distributions may be rebalanced during the training process.

Cost-Sensitive Methods

In this section, we study the cost-sensitive extensions of AdaBoost that have been introduced in the previous chapter. We first examine how cost-sensitivity can be used in order to predict the minority class more accurately in imbalanced settings and,



(a) *Banana Data,* $c = 3$ **(b)** *Cancer Data,* $c = 3$

Figure 3.5.: *Oscillating predictions of the AdaC1 algorithm on the (a) banana data and (b) cancer data.*

afterwards, contrast the methods in terms of their predictive performance regarding concurrent classification goals.

In Chapter 2 we noted that the AdaC1 method by Sun *et al.* [21] is instable in particular situations and leads to oscillating classifier output due to the exponential weight updates, which do not induce a bias towards one of the classes. We illustrated this behavior by means of a simple toy example. We now demonstrate that AdaC1 is indeed instable even when applied to real data. For this we bring in Figure 3.5, which shows precision, recall and F1-score achieved by AdaC1, which has been run on the banana and the cancer data. As can be clearly seen, precision and recall start oscillating reciprocally against each other and do not converge within 200 iterations. This confirms the results of Masnadi-Shirazi and Vasconcelos [39] and we think that this is reasons enough in order to omit further analyses of this algorithm.

Numerical Imbalance

For comparing the cost-sensitive algorithms with respect to their ability to counter imbalanced a-priori distributions of classes, we run the algorithms on each data set with different cost settings and study if they result in improved classification accuracy with regard to the minority class.

In each case (except for AdaCost), costs for the prevalent (majority) class are chosen to be 1, i.e. a "neutral" cost item for which the respective algorithm reduces to its corresponding cost-insensitive analog. False-negative costs (costs for the minority class) are scaled up within a particular interval which has been determined manually in

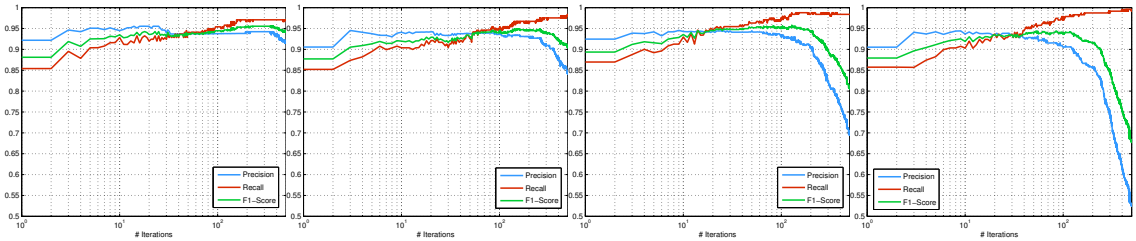
Algorithm		AdaC2			AdaC3			CSRA		
Data Set		P	R	F	P	R	F	P	R	F
Banana	y=+1	0.775	0.870	0.820	0.761	0.876	0.815	0.813	0.835	0.823
IDS	y=+1	0.958	0.874	0.909	0.946	0.889	0.912	0.972	0.871	0.911
Cancer	y=+1	0.913	0.963	0.937	0.915	0.966	0.939	0.933	0.955	**0.944**
Fire	y=+1	0.999	0.999	**0.999**	0.998	0.998	0.998	0.996	1.000	0.998
Airbag	y=+1	0.672	0.608	0.638	0.659	0.612	0.634	0.677	0.616	0.644

Algorithm		CSGA			CSLB			AdaCost		
Data Set		P	R	F	P	R	F	P	R	F
Banana	y=+1	0.804	0.852	**0.827**	0.812	0.840	0.825	0.817	0.828	0.822
IDS	y=+1	0.958	0.898	**0.922**	0.912	0.887	0.890	0.958	0.884	0.914
Cancer	y=+1	0.947	0.936	0.941	0.931	0.951	0.940	0.981	0.965	0.925
Fire	y=+1	0.998	1.000	**0.999**	0.997	1.000	0.998	0.995	1.000	0.998
Airbag	y=+1	0.704	0.614	**0.655**	0.704	0.572	0.630	0.500	0.000	0.500

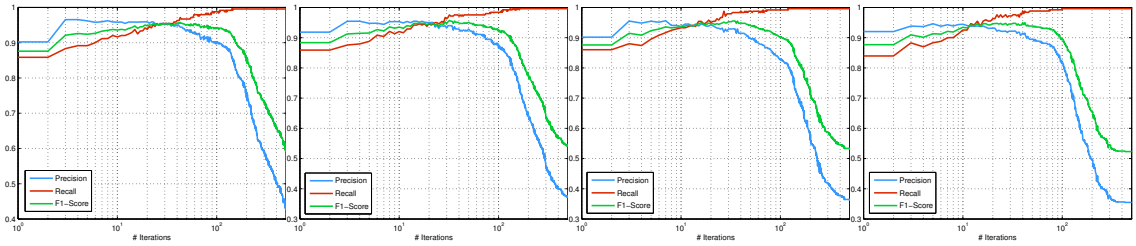
Table 3.4.: *Precision (P), Recall (R) and F1-score (F) for the positive class of cost-sensitive boosting algorithms applied to the data sets. Highest F1-scores are bolded. The values are rounded to 3 decimals. In each case the result obtained by the best parameter setting is shown.*

advance for each algorithm and data set to ensure that the parameters lie in a sound range. Since AdaCost does not reduce to original AdaBoost for any parameter assignment, it demands special treatment. In AdaCost, choices of costs are bounded to the interval $[0, 1]$, where choices close to 0 do not correspond to the intuitive meaning of "misclassifications are not penalized" since the cost adjustment function $\beta(x)$ gets constantly 0.5. Additionally, as a consequence, relative misclassification costs are upper-bounded which might inhibit determining optimal parameter settings. This seems counter-intuitive and makes it difficult to define appropriate parameter assignments. However, in order to still get an impression of AdaCost's performance we rate negative patterns (of the prevalent class) constantly with costs 0 and increment costs for positive patterns stepwise up to 1.0.

The results of experiments conducted in this context are shown in Table 3.4. The AdaC1 algorithm is omitted for the reasons mentioned above. Each of the algorithms achieves higher classification accuracy if compared with its cost-insensitive variant. Similarly to Table 3.2, the probabilistic methods outperform the "Discrete" variants slightly. Here, cost-sensitive Gentle AdaBoost (CSGA) and cost-sensitive Real AdaBoost (CSRA) share highest rankings, while CSGA achieves highest F1-scores in four of five cases and therefore does slightly better than CSRA and CSLB. This coincides with the empirical results of Li *et al.* [28].



(a) $c = 1.01$ *(b)* $c = 1.02$ *(c)* $c = 1.03$ *(d)* $c = 1.04$



(e) $c = 1.05$ *(f)* $c = 1.06$ *(g)* $c = 1.07$ *(h)* $c = 1.08$

Figure 3.6.: *Progress of precision, recall and F1-score for different cost setups of the AdaC2 algorithm applied to the cancer data.*

Figure 3.6 illustrates the effect of using costs on the classification performance regarding the positive (rare) class. Shown is the AdaC2 algorithm applied with different cost settings on the breast cancer data. In the beginning of the learning phase, precision is significantly above recall, which we identified as an effect of class imbalance in previous sections. With increasing iterations, however, rare instances gain influence on the training process and hence tend to be predicted more frequently. As a result, recall increases at the expense that more negative samples are predicted to be positive, such that precision suffers from employing costs. As costs are further increased, this effect gets even amplified and occurs earlier in time, which may have severe impact on the learning process. If costs are chosen too large, the weights assigned to positive patterns dominate the weak hypothesis and negative samples loose influence on the training process. This leads to an approximately trivial classifier that almost constantly predicts the positive class, so recall gets close to 1 (or even exactly 1) while precision and F1-score drops down. These observations indicate that costs indeed are able to alleviate the imbalance of class distributions. However, as can be seen in Figure 3.6, the methods actually are very sensitive to changes in costs and therefore require careful tweaking of parameters.

Concurrent Classification Goals

In the previous section we studied several cost-sensitive boosting algorithms in terms of their ability to alleviate the problem of numerically imbalanced class distributions, which typically causes a classifier to be biased towards the prevalent class. As mentioned in Chapter 1, imbalance may also occur as a kind of preference for one of the classes. Even if a given problem is not intrinsically imbalanced, this kind of uneven misclassification cost arises in almost every practical application. In this section we analyze the cost-sensitive boosting approaches with respect to their applicability to this second kind of imbalance.

We propose the following experimental setup. As depicted in Table 3.1, there is one class in each experiment, which constitutes the "more important" one. In the breast cancer data, for instance, the "malignant" class − which at the same time is the rare class − is the more important one, since in breast cancer detection, we prefer false alarms over misses. At the same time, however, the number of false-positive predictions should be minimized. In order to assess how well a particular algorithm can be employed to accomplish such goals of classification, we determine a parameter setting for each method, which complies the classification goal, which is given by an a-priori value in terms of precision and recall. As an example, consider again the cancer detection data. Since we have no data for testing available, we perform 5-fold cross validation over 10 runs and average the results. For a parameter assignment to meet a classification goal "at least 95% of malignant cases must be correctly predicted", the respective classifier must achieve an average recall of $95\% \pm \epsilon$ over *all* runs and *all* folds. Having accomplished this goal, the classifier that achieves lower misclassification error is regarded to be superior. We have set $\epsilon = 10^{-3}$.

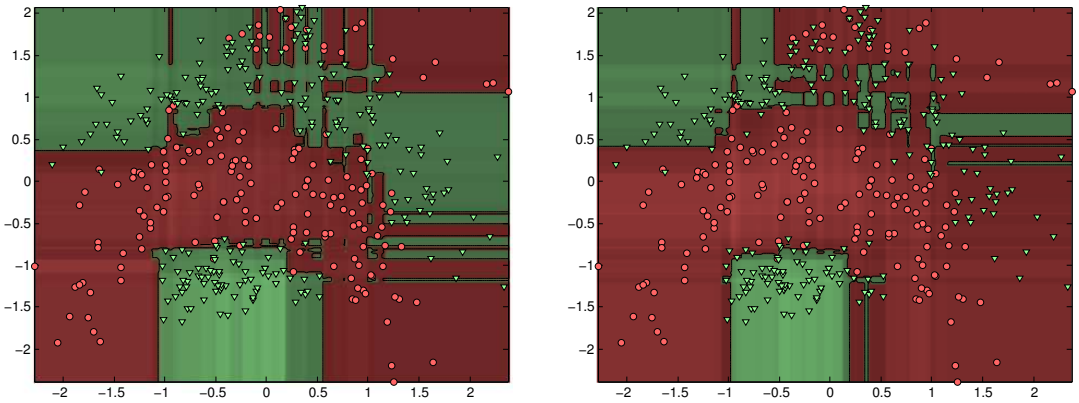
Since the banana are synthetically generated data, the class labels have no particular semantics, thus we define the positive label $y = +1$ to be the more important one in this case. Furthermore, we define 3 classification goals that lie above the recall values achieved by the cost-insensitive methods, i.e. 85%, 90% and 95%. For the fire, the airbag and the IDS data we choose only the 100% precision goal, since these data are already classified with high accuracy by the original boosting variants.

Figure 3.7 shows the decision boundaries learned by the CSRA algorithm on the banana data set in order to achieve the goals of 85% and 95% recall. As can be seen, the areas of positive (red) classifier output increasingly grow together so the risk of misclassifying a positive sample is minimized. Table 3.5 shows the results of this experiment. Although each algorithm succeeds in achieving the demanded goal (except for the IDS data), it is hard to determine an algorithm that clearly outperforms the others. From Table 3.5 it can be seen that in some cases, such as in the airbag and fire detection data, recall dramatically suffers from the high precision requirement.

Algorithm / Data Set		AdaC2			AdaC3			CSRA		
		P	R	F	P	R	F	P	R	F
Banana	R=85%	0.802	0.850	0.825	0.811	0.85	0.796	0.793	0.850	0.820
	R=90%	0.749	0.900	0.818	0.724	0.90	0.782	0.758	0.901	**0.823**
	R=95%	0.683	0.950	**0.794**	0.689	0.95	0.761	0.668	0.950	0.783
IDS	R=100%	-	-	-	-	-	-	-	-	-
Cancer	R=95%	0.897	0.950	0.923	0.892	0.949	0.920	0.922	0.950	0.935
	R=100%	0.350	1.000	0.519	0.369	1.000	0.539	0.396	1.000	0.567
Fire	P=100%	1.000	0.617	0.763	1.000	0.524	0.688	1.000	0.779	**0.872**
Airbag	P=100%	1.000	0.242	0.387	1.000	0.235	0.381	1.000	0.247	0.393

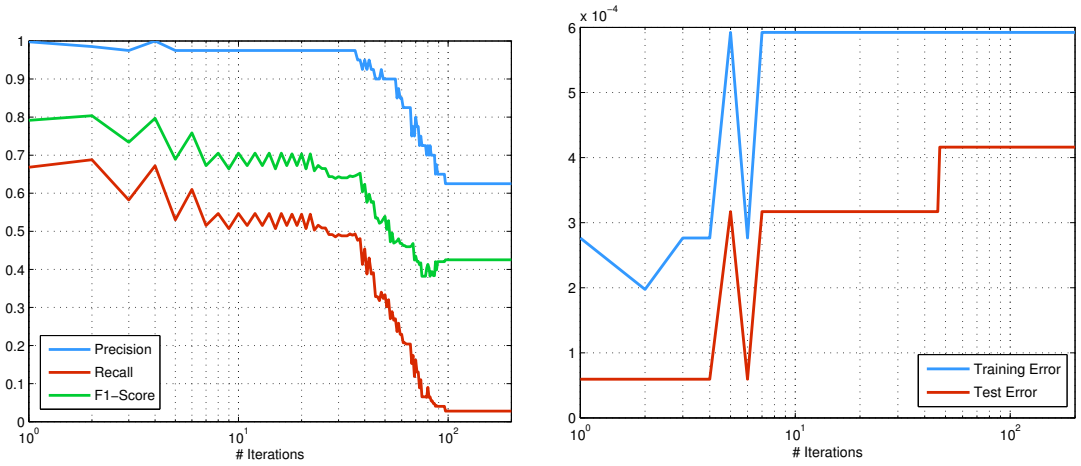
Algorithm / Data Set		CSGA			CSLB		
		P	R	F	P	R	F
Banana	R=85%	0.807	0.850	**0.828**	0.794	0.850	0.822
	R=90%	0.741	0.900	0.813	0.747	0.901	0.817
	R=95%	0.677	0.950	0.790	0.680	0.950	0.793
IDS	R=100%	-	-	-	-	-	-
Cancer	R=95%	0.921	0.950	0.935	0.923	0.951	**0.937**
	R=100%	0.401	1.000	**0.572**	0.389	1.000	0.558
Fire	P=100%	1.000	0.537	0.689	1.000	0.607	0.752
Airbag	P=100%	1.000	0.217	0.352	1.000	0.265	**0.416**

Table 3.5.: *Precision (P), Recall (R) and F1-score (F) for the positive class of cost-sensitive boosting algorithms applied to the data sets and different classification goals. Highest F1-scores are bolded. The values are rounded to 3 decimals. In each case the result obtained by the best parameter setting is shown.*



(a) *R=85%, c=1.0125* ***(b)*** *R=95%, c=1.0515*

Figure 3.7.: *Decision boundaries for CSRA applied to the banana data for achieving the 85% and 95% recall goals.*



(a) *Precision, Recall, F1-Score* ***(b)*** *Error Rates*

Figure 3.8.: *Progress of Precision, Recall, F1-Score and Error Rate of the CSGA algorithm applied to IDS data.*

In the IDS data however, none of the algorithms succeeded in achieving the goal of 100% precision, i.e. no erroneously refused network connections. Indeed, an interesting effect can be observed when costs are chosen too high. In Figure 3.8 we show the progress of precision, recall and F1-score as well as the training and test error rates produced by the CSGA algorithm applied to the IDS data, having set costs for false-positive predictions as $c = 10$.

As can be seen, precision and recall rapidly (even exponentially) drop down after 35 iterations. Up to the completion of this thesis, we have no definite theoretical explanation for this behavior, but we think that it might be related to the exponential gain of influence of misclassified positive samples, which, at some point in time, overwhelm the weights of correctly predicted negative samples, though amplified by a constant cost factor. This might lead to distortions in the training process.

Our results imply that, on the one hand, costs can be used in order achieve a certain error rate regarding a particular class. However, we can conclude that this goal is not guaranteed to be reachable by scaling up misclassification costs of the important class. Indeed, each algorithm succeeded in a second run, when costs of the less important class have been scaled down. Though intuition might suggest, that, if we define $c = \infty$, i.e. infinite misclassification costs for one class, we might obtain at least a trivial classifier for this particular class, in practice, this is not necessarily the case. As one reason for that we note that the weights do only have impact on the *individual* classifiers and *not* on the ensemble. That is, if exclusively the weight updates are modified in order to assign a particular class higher weights, this does not necessarily induce a bias towards this class on the ensemble. As a trivial example,

consider a binary classification problem comprising two points x_1 and x_2, $x_1 \neq x_2$, of different classes and we wish to build a prediction model that never fails on one of the two. In terms of boosting, costs for either of the two classes can be set arbitrarily high, but they will not induce any bias to the ensemble decision. However, setting costs $c = 0$ for one class will accomplish the goal. This might be another reason why introducing costs does not necessarily lead to arbitrarily high recall of a particular class. Hence, this experiment supports the common finding that under-sampling the less important class is more effective than over-sampling the more important one.

Summary

We conclude this chapter by giving a concise valuation of each algorithm in different scenarios, regarding our theoretical and empirical findings. We characterize the respective algorithms in four categories: the regularization behavior or robustness against overfitting, respectively, the ability for coping with numerical imbalance and classification performance with regard to class preferences or uneven misclassification costs. Additionally, we assign each algorithm a valuation scoring its predictive performance in general, where ▲ means "good" and ▼ means "poor".

The algorithms that are regarded as not being promising due to empirical results or theoretical analysis are labeled ▼▼▼ . The CSB0-2 algorithms have been ruled out due to poor performance in early experiments and lack of theoretical motivation, as described in Chapter 2. Likewise, the AdaC1 algorithm has been proven as being instable for particular cost settings in this chapter.

We found that AdaCost achieves only very poor performance improvements compared to other cost-sensitive methods, and suffers from counter-intuitive semantics of parameters, which complicate its practical application. We therefore regard AdaCost to be inferior to the other algorithms.

In almost all of our experiments, the probabilistic variants outperform their "discrete" analogs. At the head GentleBoost, which robustly yields good classification results and has a powerful cost-sensitive extension.

WeightBoost, as a regularized boosting method significantly outperforms AdaBoost$_{Reg}$ which dramatically suffers from the imbalance problem, and even exceeded the cost-sensitive variants in some of our experiments.

Algorithm	**Criterion**			
	Predictive Power	Regularization	Imbalance	Concurrent Goals
(Discrete) AdaBoost	▲	▼	▼	▼
CSB0, CSB1, CSB2	▼▼▼	▼▼▼	▼▼▼	▼▼▼
AdaC1	▼▼▼	▼▼▼	▼▼▼	▼▼▼
AdaC2	▲	▼	▲	▲
AdaC3	▲▲	▼	▲▲	▲
AdaCost	▼▼▼	▼▼▼	▼▼▼	▼▼▼
RareBoost	▼	▼	▼	▼
AdaBoost_{Reg}	▲	▲	▼	▼
WeightBoost	▲	▲▲▲	▲	▼
RealBoost	▲▲	▼	▼	▼
GentleBoost	▲▲▲	▼	▼	▼
LogitBoost	▲▲	▼	▼	▼
CSRA	▲	▼	▲▲	▲
CSGA	▲	▼	▲▲▲	▲
CSLB	▲	▼	▲	▲

Table 3.6.: *Overview of boosting algorithms considered in Chapter 3.*

Chapter 4

Implementation

In context of this thesis a Matlab toolbox has been developed which implements all of the 14 boosting algorithms that have been discussed in the previous chapters. In this chapter we briefly outline the implementation and usage of this library.

4.1. Base Classifiers

The following base classifiers have been implemented:

- *Decision stump*: A Decision stump is a linear discriminant classifier that subdivides the input space into cuboid regions, i.e. its decision boundary corresponds to a parallel of one coordinate axis. The dimension and variable for each stump is chosen, such that the weighted training error is minimized. Therefore, a stump can be regarded as a CART that has only one single terminal node. Our stump implementation returns an estimate of the weighted probability $P_w(y = 1 \mid x)$. If regression is the goal instead of classification (e.g. with LogitBoost), the output is given by a weighted average over all samples in a terminal node.
- *CART*: Classification and Regression Trees can be regarded as the decision stump algorithm being recursively applied to the data. We use the built-in Matlab implementation of CART, which returns a weighted probability estimate $P_w(y = 1 \mid x)$.
- *SVM*: First trials with Support-Vector-Machines have been conducted using libSVM [77], but since it is hard to obtain reliable probability estimates from SVM models, the usage of SVM stubs in the probabilistic boosting versions is not recommended. However, SVM stubs can be combined with Discrete AdaBoost, where the base learners only have to provide binary class labels.

4.2. Usage

We briefly describe how the Matlab toolbox is used. The main functionality is embodied within two functions: `boost_learn` and `boost_eval`. The signature and parameters are defined as follows.

```
boost = boost_learn(X,Y,method, baselearner, M, par1, val1, ...);
```

which returns a `boost` structure representing the prediction model that has been learned. `X` is an $N \times p$ matrix holding N observations of p variables. Y is a column vector holding the class labels. Parameters:

`method` − The boosting algorithm to be used. Possible values:

- `'ada_boost'` − (Discrete) AdaBoost
- `'real_boost'` − RealBoost
- `'gentle_boost'` − GentleBoost
- `'logit_boost'` − LogitBoost
- `'ada_c1'` − AdaC1
- `'ada_c2'` − AdaC2
- `'ada_c3'` − AdaC3
- `'ada_cost'` − AdaCost
- `'cs_realboost'` − Cost-Sensitive Real AdaBoost (CSRA)
- `'cs_gentleboost'` − CostSensitive Gentle AdaBoost (CSGA)
- `'cs_logitboost'` − CostSensitive LogitBoost (CSLB)
- `'rare_boost'` − RareBoost
- `'adaboost_reg'` − AdaBoost$_{Reg}$
- `'weight_boost'` − WeightBoost

`M` − The number of iterations.

`baselearner` − The base learning algorithm to be used. Currently allowed values are

- `'stump'` − Decision stump.
- `'cart'` − Classification and Regression Trees (CART) [47].
- `'svm'` − Support-Vector-Machine (SVM) [51].

`par1, val1` − Parameter-value pairs. Options for the learning process:

- `'c'` − Cost parameter in case of cost-sensitive methods or regularization parameter in case of regularized methods, respectively.
- `'cv'` − Performs K-fold cross-validation by stratified sampling. The result is written to the console. Default is 1 (no validation).
- `'testdata'` − A test data set X and labels Y. The result is written to the console.
- `'plot'` − Plots a figure of the progression of training and test error and precision, recall and F1-score over the iterations. Averaged over all folds in case of cross validation. Default is `'no'`.
- `'boundary'` − In case of 2-dimensional problems, plots the decision surface. Default is `'no'`.
- `'v'` − Verbose mode. Default is `'no'`.

```
Y = boost_eval(boost, X, Y);
```

Returns the predictions, i.e. a column vector Y holding the responses $F(x)$ of the ensemble.

`boost` − A predictor model learned by the `boost_learn` procedure.

`X` − An $N \times p$ matrix holding the observations, where each row is a data point and each column represents a variable.

`Y` − [optional] Column vector holding class labels. If true class labels are provided, the corresponding error, precision, recall and F1-score are written to the console.

Optionally, there is a function `show_decision_boundary(boost,X,Y)`, which can be used independently in order to visualize the output of a boosting classifier applied to a a two dimensional problem.

Example

Let us demonstrate the usage of our Matlab toolbox for boosting by means of a simple example. The following code generates two slightly overlapping gaussian clusters of two classes, and applies the original Discrete AdaBoost algorithm, using decision stumps as base learners.

```
N = 1000;
```

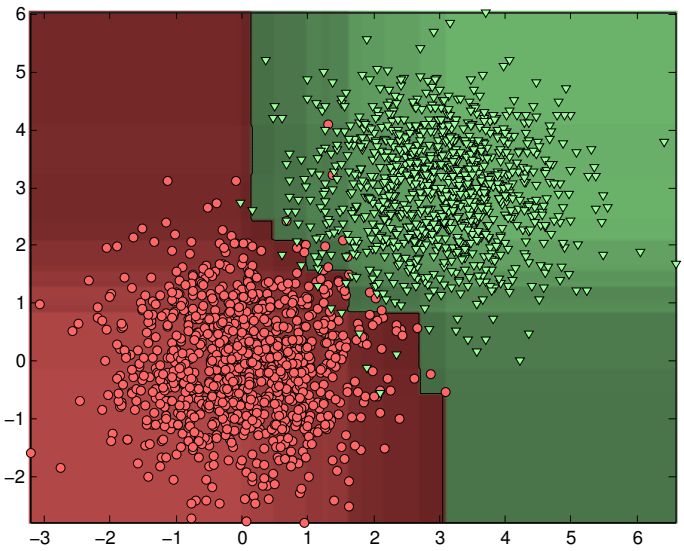


Figure 4.1.: *Decision boundary of the code example.*

```
M = 50;
X=randn(N,2);
X=[X;randn(N,2)+3];
Y(1:N,1) = 1;
Y(N+1:2*N,1) = -1;
boost = boost_learn(X,Y,'adaboost','stump',M);
```

A call of

```
show_decision_boundary(boost,X,Y);
```

displays the corresponding scatter plot shown in Figure 4.1.

Chapter 5

Conclusions

The presence of class imbalance in machine learning, i.e. learning from rare or skewed classes, and/or the presence of uneven costs of making different kinds of errors represents a challenging topic for the machine learning community, which attracted a great deal of attention during the recent decade. Traditional standard classifier learning algorithms typically assume balanced class distributions. One of the most popular approaches in the area of ensemble learning is boosting. Boosting has been proven as being an efficient and powerful meta-algorithm for converting a weak classifier, which has only poor individual predictive performance, into a strong classifier with arbitrarily low training error. However, the original boosting algorithms have severe drawbacks in imbalanced problems as they only consider the overall accuracy, which is not class-specific.

This thesis investigates boosting algorithms for learning in the presence of imbalanced classes and uneven misclassification costs. In particular, we have addressed the *AdaBoost* algorithm by Freund and Schapire [6]. A large number of extensions to AdaBoost have been proposed in literature in order to tackle the imbalance problem. Among these, introducing cost items in the error function is the prevalent approach.

We have discussed the cost-sensitive extensions of AdaBoost theoretically and have approved our findings by empirical evaluation. We identified the *AdaC1* algorithm by Sun *et al.* [21] as being instable, and we theoretically have shown that introducing exponential cost items into the weight updates is not recommended. We found that exponential cost setups, in contrast to multiplicative ones, fail in inducing an effective bias towards the respective class and hence such approaches seem not to be promising. We have confirmed the findings of previous empirical studies, that one of the earliest approaches in cost-sensitive boosting, namely *AdaCost*, yields only suboptimal classification performance. We additionally gave a theoretical explanation for this.

A boosting approach countering skewed classes, that abstains from using explicit cost items, *RareBoost* [32], has been exposed as being a special case of the well-known *RealBoost* algorithm by Friedman *et al.* [11]. We provide proofs for RareBoost being even identical to RealBoost if decision stumps are used as base classifiers, and for disproving the common presumption that RareBoost is instable in particular situations.

We have studied the behavior of the AdaBoost algorithm in presence of noise and overlapping class distributions, and confirmed its tendency to overfit, which has been undetected for a considerable time after AdaBoost's inception, and we have reviewed the common assumption that this is due to its particularly aggressive strategy towards misclassifications during the learning phase. We found that a *soft-margin* solution $AdaBoost_{Reg}$ by Rätsch [37] particularly suffers from imbalance since it assigns equal "mistrust" to patterns of both classes, further reducing the influence of the minority class. The *WeightBoost* algorithm by Liu *et al.* [38] is a kind of *mixture-of-experts* system, which succeeds in rebalancing class distributions by removing patterns that are easy to classify permanently from the training set.

We have approved our theoretical findings by an empirical study using several data sets from real-world applications as well as publicly available benchmark data. The algorithms under consideration have been discussed and compared in terms of their ability to improve predictive performance in the presence of rare classes and tackling the problem of uneven misclassification costs. Our empirical findings are summarized in Table 3.6, which gives a qualitative assessment of each algorithm considered in this work. We found that algorithms inducing a bias towards the important class by means of a multiplicative cost factor in the weight update scheme, such as *AdaC2, CSRA, CSGA* and *CSLB*, significantly outperform their competitors. Furthermore, a tendency could be detected that the probabilistic variants of boosting are superior to their discrete analogs. The experiments using highly skewed IDS (Intrusion Detection System) data show that assigning higher costs to the more important class cannot be used for implementing uneven misclassification costs in general. Instead, scaling costs down for the less important class yields better results. Since the reweighting mechanism of AdaBoost can be regarded as a smooth form of resampling, these results support the common finding that under-sampling is superior to over-sampling [13, 16, 17].

As a subject for future investigations combining the concept of soft-margin regularization and cost-sensitivity appears to be an attractive line of research. As pointed out, we identified the AdaBoost_{Reg} algorithm as being sensitive to class imbalance since its mistrust estimate treats patterns of both classes equally. Here, the mistrust estimate should differentiate between the classes.

In this work, only binary classification problems have been considered. Extensions to multiclass problems remain an open issue. For the original AdaBoost procedure such extensions exist, but have been only rarely discussed in context of cost-sensitivity and imbalance [21, 33].

Appendix A
Proofs

In this Appendix proofs are provided of lemmata that can be found in literature. However, in most sources the proofs are kept very tight, so we think it is expedient to supply the reader with more detailed reformulations of these for better re-enacting the concepts presented in this work.

Lemma 4 *The normalization factor*

$$Z_m = \sum_{n=1}^{N} w_m^{(n)} e^{-y_n \alpha_m f_m(x_n)}$$

is minimized at

$$\alpha_m = \frac{1}{2} \ln \frac{1-\epsilon_m}{\epsilon_m}, \qquad \textit{with } \epsilon_m = \sum_{n=1}^{N} w_m^{(n)} \mathbf{1}_{y_n \neq f_m(x_n)}.$$

Proof

$$\begin{aligned} Z_m = \sum_{n=1}^{N} w_m^{(n)} e^{-y_n \alpha_m f_m(x_n)} &= \sum_{m=1}^{M} w_m^{(n)} \left(\frac{1+y_n f_m(x_n)}{2} e^{-\alpha_m} + \frac{1-y_n f_m(x_n)}{2} e^{\alpha_m} \right) \\ &= \frac{1}{2} e^{-\alpha_m} \sum_{m=1}^{M} w_m^{(n)} \left(1 + y_n f_m(x_n)\right) \\ &\qquad + \frac{1}{2} e^{\alpha_m} \sum_{m=1}^{M} w_m^{(n)} \left(1 - y_n f_m(x_n)\right) \end{aligned}$$

Setting the derivative w.r.t. α_m to zero yields

$$\begin{aligned}
\frac{1}{2}e^{\alpha_m}\sum_{m=1}^{M} w_m^{(n)}\left(1-y_n f_m(x_n)\right) &= \frac{1}{2}e^{-\alpha_m}\sum_{m=1}^{M} w_m^{(n)}\left(1+y_n f_m(x_n)\right) \\
\alpha_m + \ln\sum_{m=1}^{M} w_m^{(n)}\left(1-y_n f_m(x_n)\right) &= -\alpha_m \ln\sum_{m=1}^{M} w_m^{(n)}\left(1+y_n f_m(x_n)\right) \\
2\alpha_m &= \ln\sum_{m=1}^{M} w_m^{(n)}\left(1+y_n f_m(x_n)\right) \\
&\qquad - \ln\sum_{m=1}^{M} w_m^{(n)}\left(1-y_n f_m(x_n)\right) \\
\alpha_m &= \frac{1}{2}\ln\frac{\sum_{m=1}^{M} w_m^{(n)}\left(1+y_n f_m(x_n)\right)}{\sum_{m=1}^{M} w_m^{(n)}\left(1-y_n f_m(x_n)\right)} \\
\alpha_m &= \frac{1}{2}\ln\frac{1+\sum_{m=1}^{M} w_m^{(n)} y_n f_m(x_n)}{1-\sum_{m=1}^{M} w_m^{(n)} y_n f_m(x_n)} \\
\alpha_m &= \frac{1}{2}\ln\frac{1-\epsilon_m}{\epsilon_m}, \qquad \text{with } \epsilon_m = \sum_{n=1}^{N} w_m^{(n)} \mathbf{1}_{y_n \neq f_m(x_n)}
\end{aligned}$$

□

Lemma 5 *A Newton-like step towards the minimum of the objective function*

$$J(F_{m-1}(x)+\alpha_m f_m(x)) = \mathbb{E}\left(e^{-y(F_{m-1}(x)+\alpha_m f_m(x))}\,|\,x\right)$$

is given by

$$f_m(x) = \begin{cases} +1 & \text{if } P_w(y=+1\,|\,x) \geq P_w(y=-1\,|\,x) \\ -1 & \text{if } P_w(y=+1\,|\,x) < P_w(y=-1\,|\,x) \end{cases}.$$

Proof We follow the proof of Friedman *et al.* [11]. A Newton update step towards the minimum of a convex objective function $J(x)$ is given by minimization of the second order Taylor expansion of $J(x)$ around the current estimate of its minimum x_t:

$$x_{t+1} = x_t + \arg\min_{\Delta x}\left(J(x_t) + J'(x_t)\Delta x + \frac{1}{2}J''(x_t)\Delta x^2\right). \tag{A.1}$$

In our case, $e^{-yF(x)}$ constitutes the objective function and the current estimate is given by $F_{m-1}(x)$. For fixed α_m, expanding to second order around $f_m(x) = 0$ yields

$$\begin{aligned} J(F_{m-1}(x) + \alpha_m f_m(x)) &= \mathbb{E}\left(e^{-y(F_{m-1}(x)+\alpha_m f_m(x))} \,|\, x\right) \\ &\approx \mathbb{E}\left(e^{-yF_{m-1}(x)}\left(1 - y\alpha_m f_m(x) + y^2\alpha_m^2 f_m(x)^2/2 \,|\, x\right)\right) \\ &= \mathbb{E}\left(e^{-yF_{m-1}(x)}(1 - y\alpha_m f_m(x) + \alpha_m^2/2) \,|\, x\right), \end{aligned}$$

since $y^2 = 1$ and $f_m(x)^2 = 1$, such that the "Δ" to be added to $F_{m-1}(x)$ is given by

$$f_m(x) = \arg\min_f \mathbb{E}_w\left(1 - y\alpha_m f(x) + \alpha_m^2/2 \,|\, x\right), \tag{A.2}$$

where we have defined $w = w(x, y) = e^{-yF_{m-1}(x)}$ and the notation $\mathbb{E}_w(\cdot \,|\, x)$ refers to the *weighted conditional expectation*, which is defined as

$$\mathbb{E}_w(g(x, y) \,|\, x) = \frac{\mathbb{E}(w(x, y)g(x, y) \,|\, x)}{\mathbb{E}(w(x, y) \,|\, x)}. \tag{A.3}$$

Since all other terms in (A.2) are constant, for any $\alpha_m > 0$, (A.2) is minimized when

$$\mathbb{E}_w(y f_m(x) \,|\, x) = f_m(x) \cdot P_w(y = 1 \,|\, x) - f_m(x) \cdot P_w(y = -1 \,|\, x)$$

is maximized. This is the case if $f_m(x)$ meets

$$f_m(x) = \begin{cases} +1 & \text{if } P_w(y = +1 \,|\, x) \geq P_w(y = -1 \,|\, x) \\ -1 & \text{if } P_w(y = +1 \,|\, x) < P_w(y = -1 \,|\, x) \end{cases}.$$

□

Lemma 6 *The objective function*

$$J\left(F_{m-1}(x) + f_m(x)\right) = \mathbb{E}\left(e^{-y(F_{m-1}(x)+f_m(x))} \,|\, x\right)$$

is minimized at

$$f_m(x) = \frac{1}{2} \ln \frac{P_w(y = +1 \,|\, x)}{P_w(y = -1 \,|\, x)}.$$

Proof Following Friedman *et al.* [11]:

$$\begin{aligned} J\left(F_{m-1}(x) + f_m(x)\right) &= \mathbb{E}\left(e^{-y(F_{m-1}(x)+f_m(x))} \,|\, x\right) \\ &= \mathbb{E}\left(e^{-yF_{m-1}(x)} e^{-yf_m(x)} \,|\, x\right) \\ &= \mathbb{E}\left(e^{-yF_{m-1}(x)} \mathbf{1}_{y=1} e^{-f_m(x)} + e^{-yF_{m-1}(x)} \mathbf{1}_{y=-1} e^{f_m(x)} \,|\, x\right) \\ &= e^{-f_m(x)} \mathbb{E}\left(e^{-yF_{m-1}(x)} \mathbf{1}_{y=1} \,|\, x\right) + e^{f_m(x)} \mathbb{E}\left(e^{-yF_{m-1}(x)} \mathbf{1}_{y=-1} \,|\, x\right) \\ &= e^{-f_m(x)} \mathbb{E}_w\left(\mathbf{1}_{y=1} \,|\, x\right) + e^{f_m(x)} \mathbb{E}_w\left(\mathbf{1}_{y=-1} \,|\, x\right), \end{aligned}$$

where we have defined $w = w(x, y) = e^{-yF_{m-1}(x)}$, and $\mathbb{E}_w(\cdot \,|\, x)$ refers to the weighted conditional expectation defined in (2.35). Setting the derivative with respect to $f_m(x)$ to zero yields

$$\begin{aligned}
e^{f_m(x)}\mathbb{E}_w\left(\mathbf{1}_{y=-1} \,|\, x\right) &= e^{-f_m(x)}\mathbb{E}_w\left(\mathbf{1}_{y=1} \,|\, x\right) \\
f_m(x) + \ln \mathbb{E}_w\left(\mathbf{1}_{y=-1} \,|\, x\right) &= -f_m(x) + \ln \mathbb{E}_w\left(\mathbf{1}_{y=1} \,|\, x\right) \\
2 \cdot f_m(x) &= \ln \mathbb{E}_w\left(\mathbf{1}_{y=1} \,|\, x\right) - \ln \mathbb{E}_w\left(\mathbf{1}_{y=-1} \,|\, x\right) \\
f_m(x) &= \frac{1}{2} \ln \frac{\mathbb{E}_w\left(\mathbf{1}_{y=1} \,|\, x\right)}{\mathbb{E}_w\left(\mathbf{1}_{y=-1} \,|\, x\right)} \\
f_m(x) &= \frac{1}{2} \ln \frac{P_w(y = +1 \,|\, x)}{P_w(y = -1 \,|\, x)}
\end{aligned}$$

□

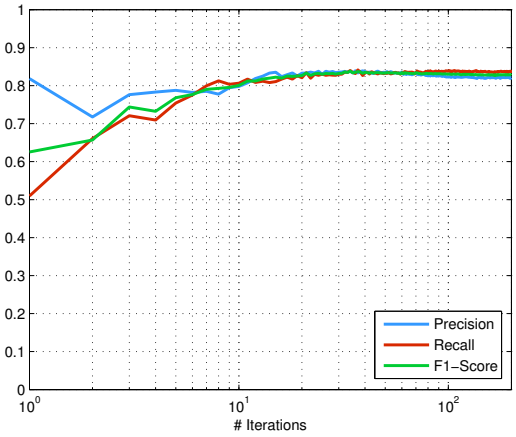
Appendix B

Evaluation Data

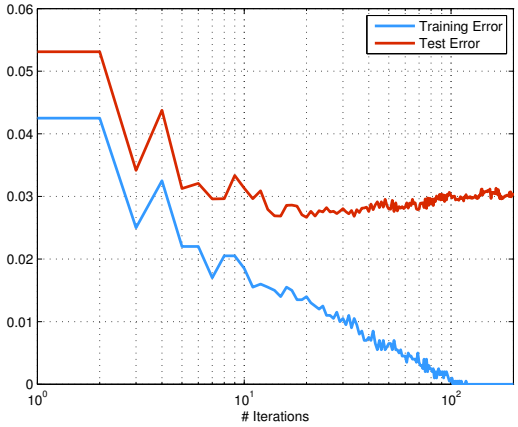
This appendix provides detailed results obtained by conducting the experiments described in Chapter 3. For a selection of algorithms and data sets, we provide the progression of precision, recall, F1-score and/or training and test error over the iterations. In case of two-dimensional data the decision boundaries of the classifier at the end of the training in combination with the training patterns are also plotted. For the algorithms driven by a parameter, results of 10 different parameter values are shown, that lie in an interval where the respective algorithm performs best (s.a. Chapter 3). An additional precision-recall-F-score (PRF) curve summarizes the performance of an algorithm regarding different parameter settings.

Banana Data Set

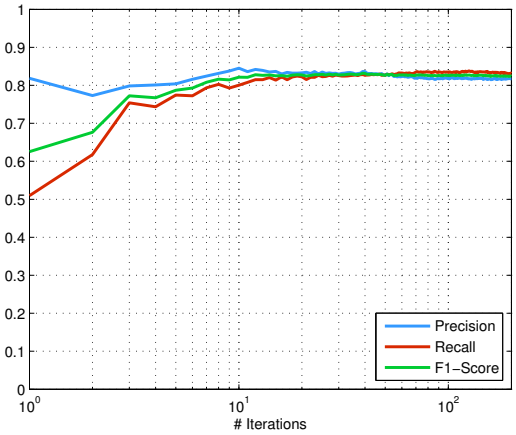
Iterations/Base Classifiers:	200 CART
Evaluation Mode:	Test Set Evaluation averaged over 10 Subsets.



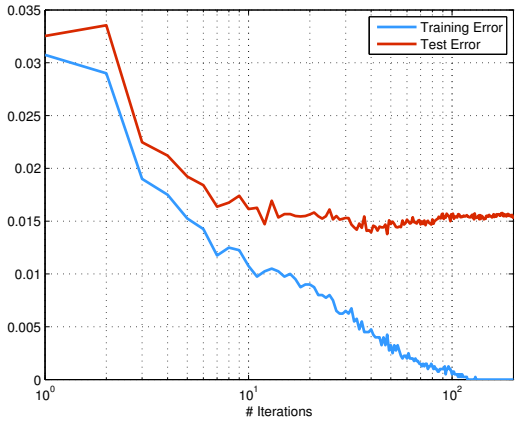
(a) AdaBoost



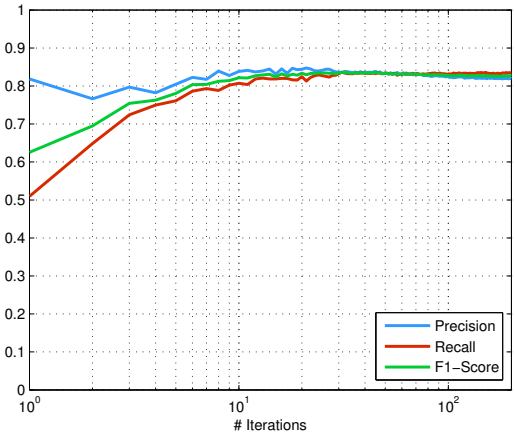
(b) AdaBoost



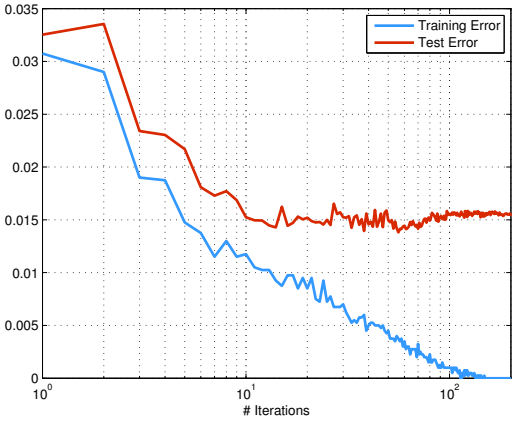
(c) RealBoost



(d) RealBoost

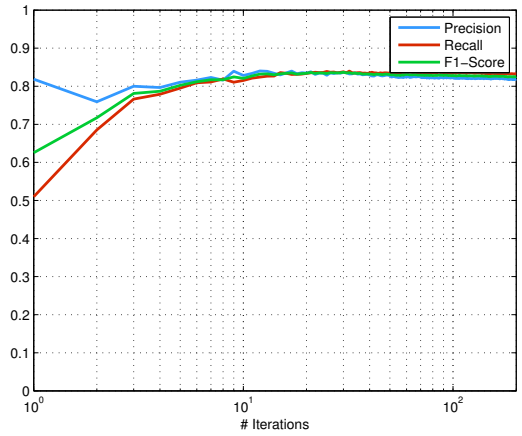


(e) GentleBoost

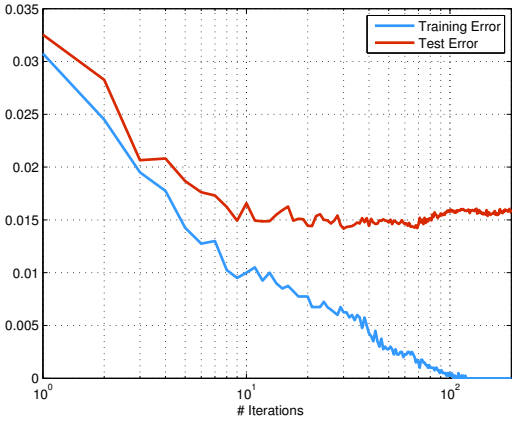


(f) GentleBoost

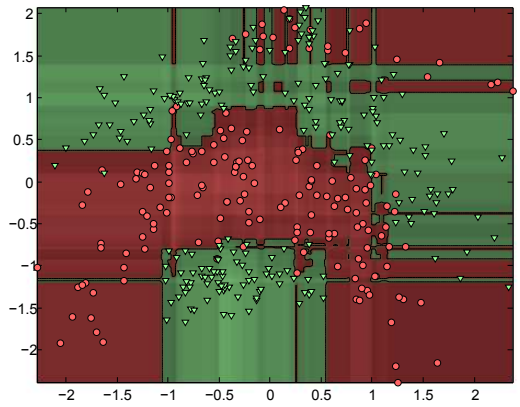
Figure B.1.: *Cost-insensitive algorithms on banana data.*



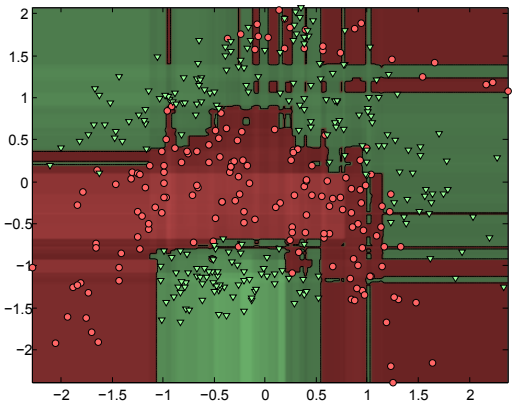
(a) LogitBoost



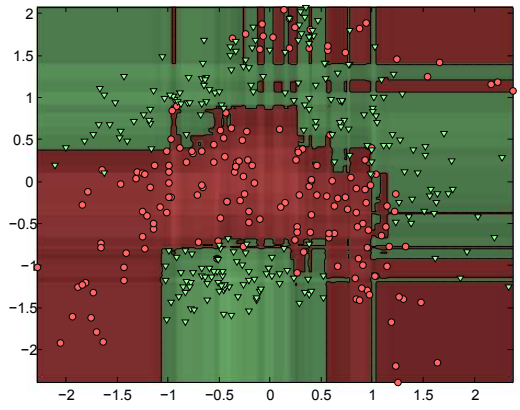
(b) LogitBoost



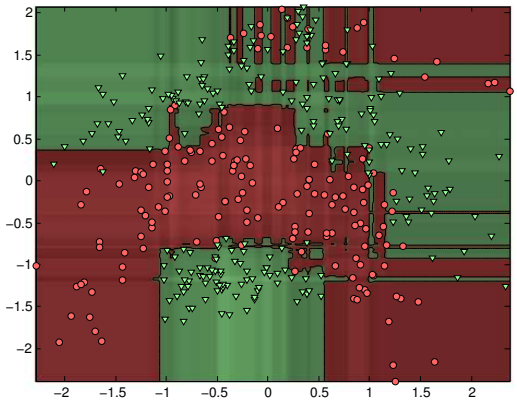
(c) AdaBoost



(d) RealBoost

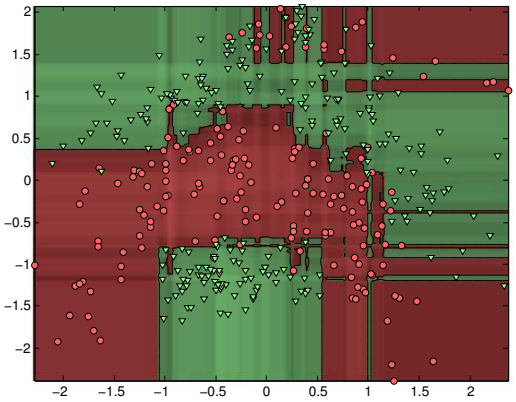


(e) GentleBoost

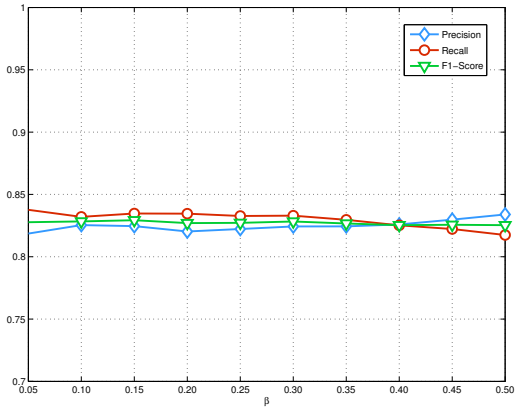


(f) LogitBoost

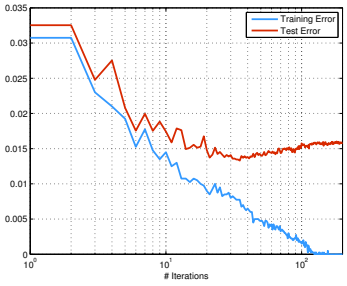
Figure B.2.: *Cost-insensitive algorithms on banana data (ctd.).*



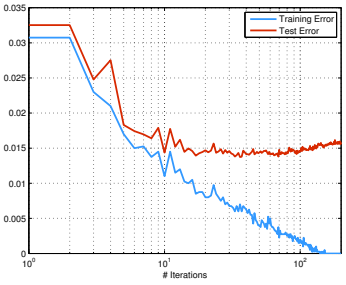
(a) Decision boundary, $\beta = 0.15$



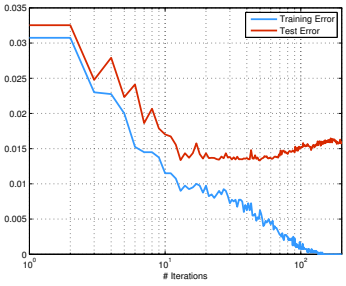
(b) PRF for different β, best at $\beta = 0.15$



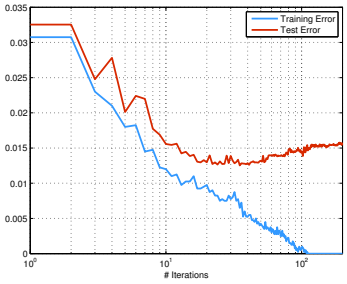
(c) $\beta = 0.05$



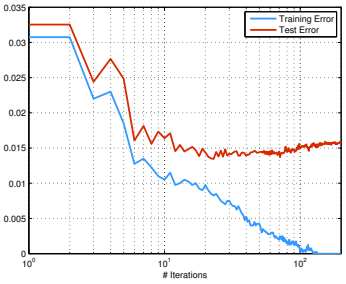
(d) $\beta = 0.10$



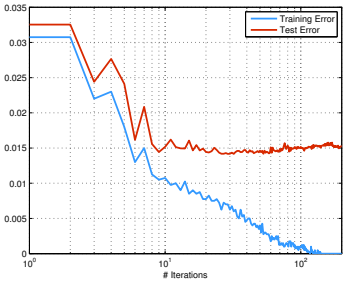
(e) $\beta = 0.15$



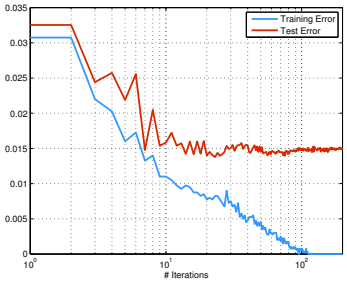
(f) $\beta = 0.20$



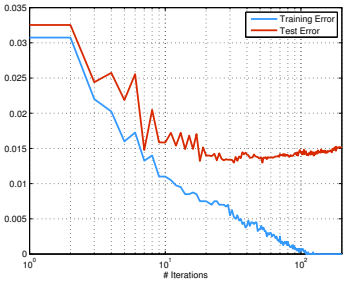
(g) $\beta = 0.25$



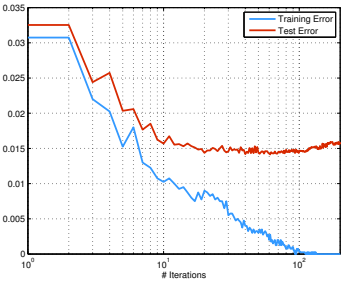
(h) $\beta = 0.30$



(i) $\beta = 0.35$

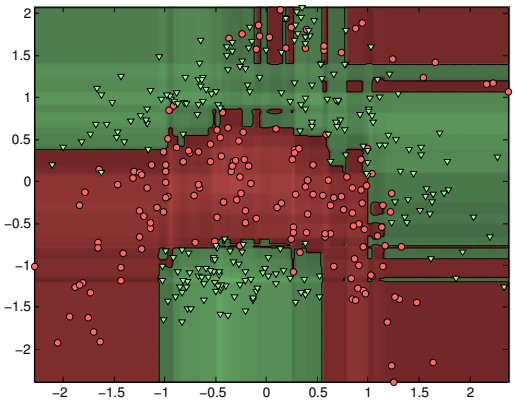


(j) $\beta = 0.40$

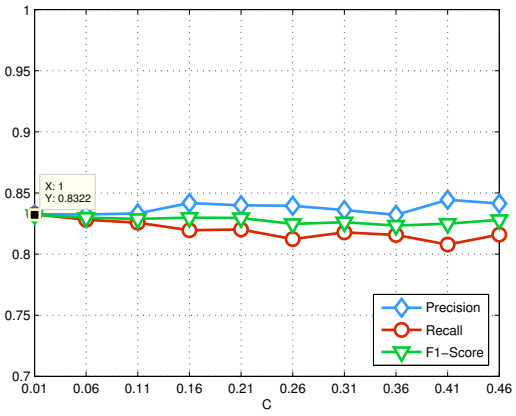


(k) $\beta = 0.45$

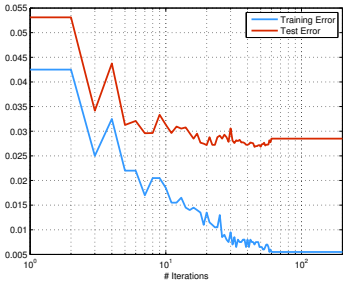
Figure B.3.: *WeightBoost on banana data.*



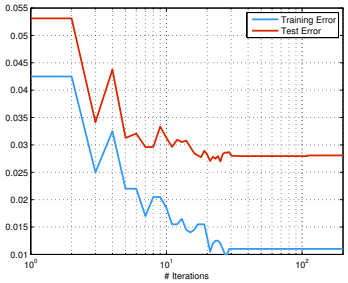
(a) Decision boundary, $C = 0.01$



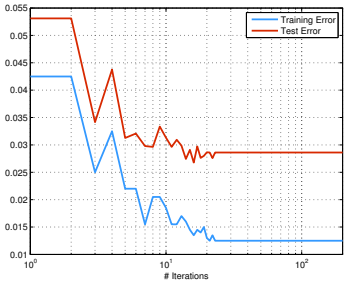
(b) PRF for different C, best at $C = 0.01$



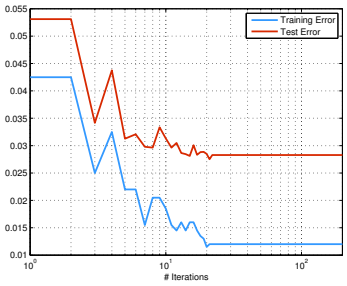
(c) $C = 0.01$



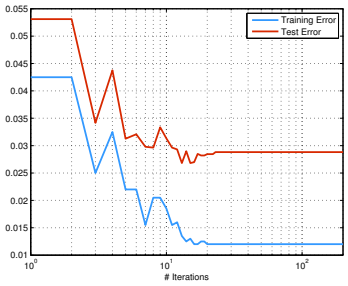
(d) $C = 0.06$



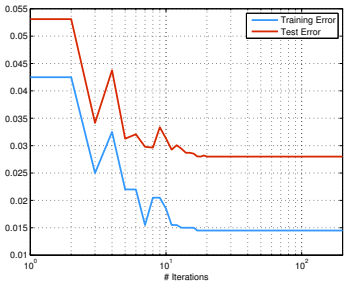
(e) $C = 0.11$



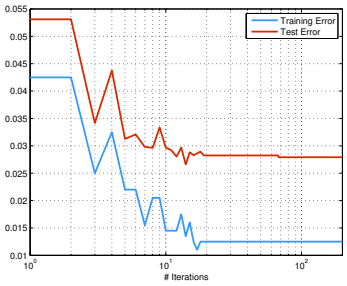
(f) $C = 0.16$



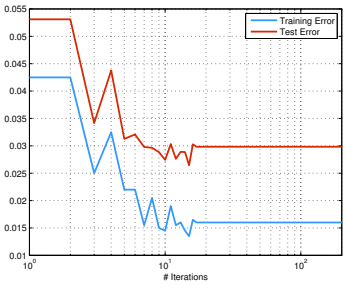
(g) $C = 0.21$



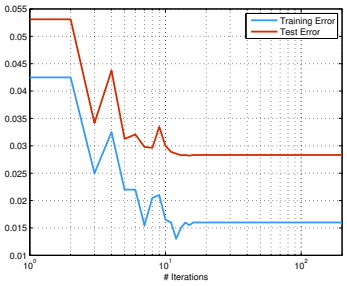
(h) $C = 0.26$



(i) $C = 0.31$

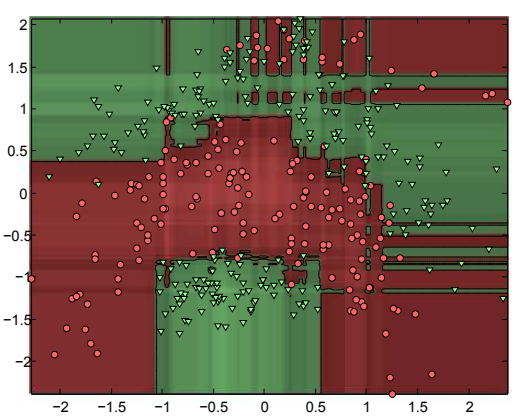


(j) $C = 0.36$

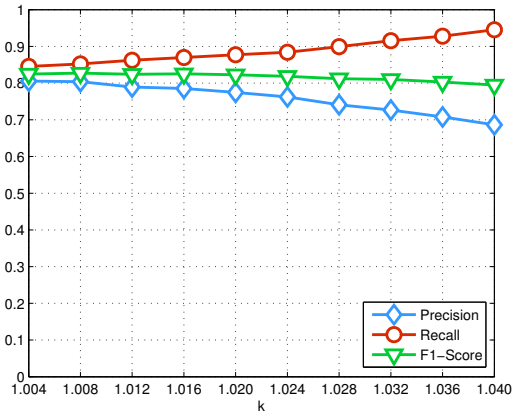


(k) $C = 0.41$

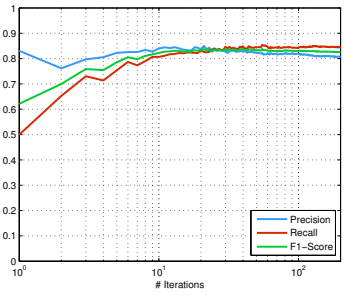
Figure B.4.: *AdaBoost_{Reg} on banana data: Due to regularization, training error stops decreasing at a particular level and test error does not increase.*



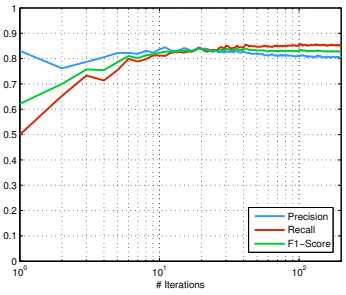
(a) Decision boundary, $C = 1.008$



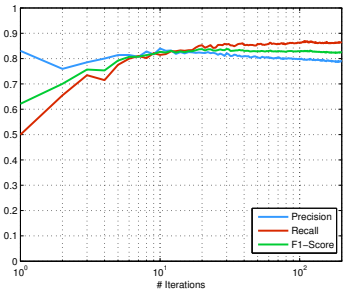
(b) PRF for different C, best at $C = 1.008$



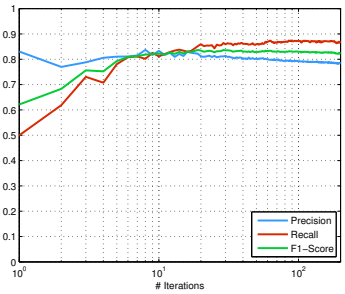
(c) $C = 1.004$



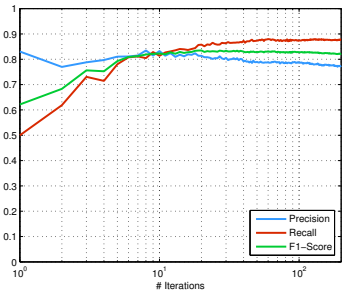
(d) $C = 1.008$



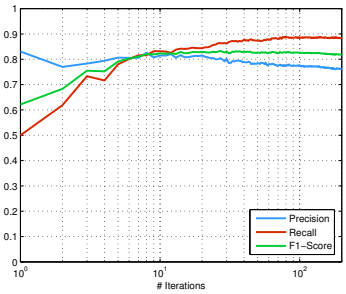
(e) $C = 1.012$



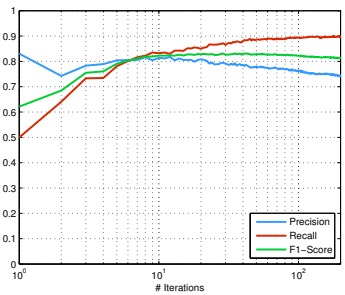
(f) $C = 1.016$



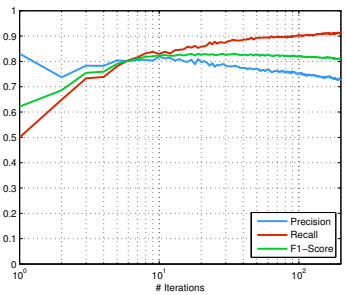
(g) $C = 1.020$



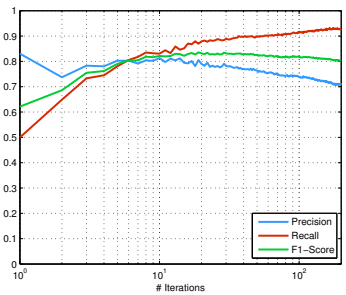
(h) $C = 1.024$



(i) $C = 1.028$



(j) $C = 1.032$

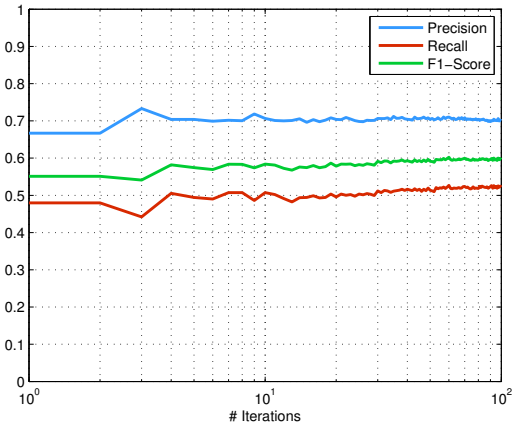


(k) $C = 1.036$

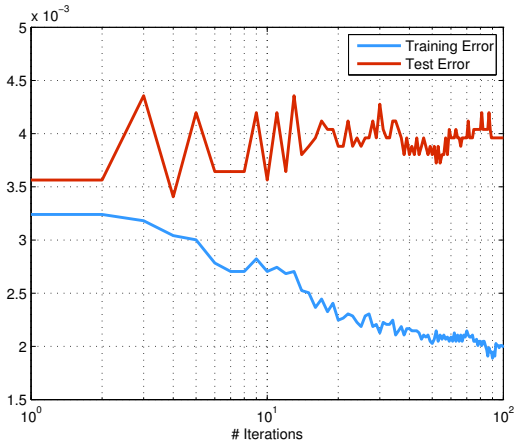
Figure B.5.: *CSGA on banana data: Recall (regarding the positive (red) class) increases with increasing costs to the expense of worse precision.*

Airbag Deployment / Crash Data

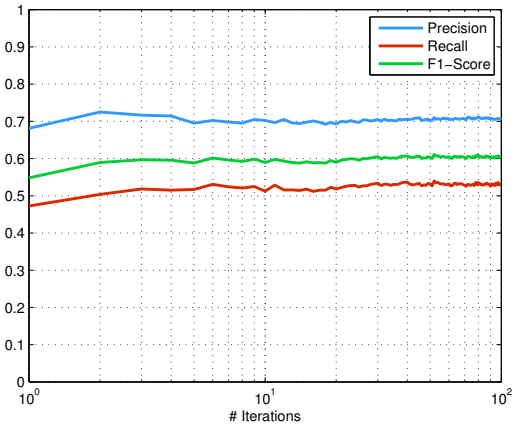
Iterations/Base Classifiers: 200 CART
Evaluation Mode: Test Set Evaluation averaged over 10 Subsets.



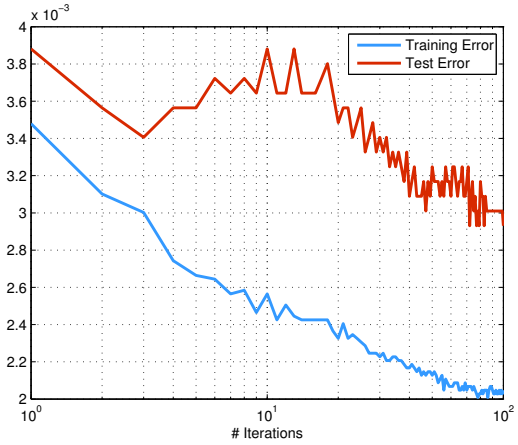
(a) AdaBoost



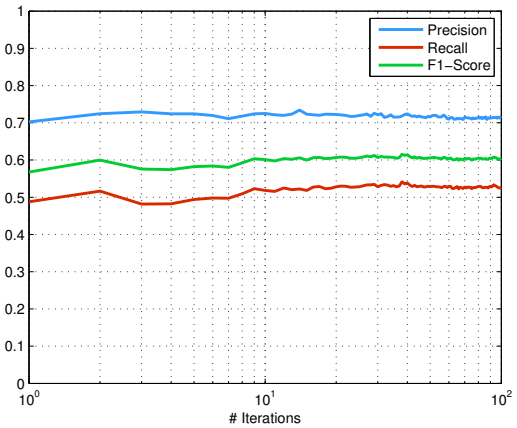
(b) AdaBoost



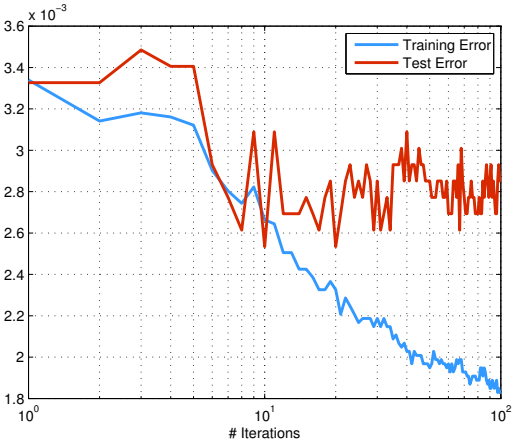
(c) RealBoost



(d) RealBoost

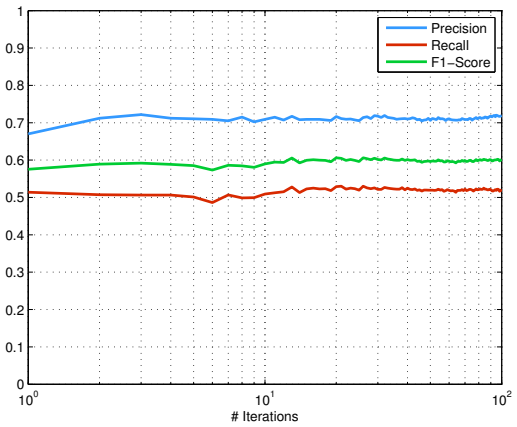


(e) GentleBoost

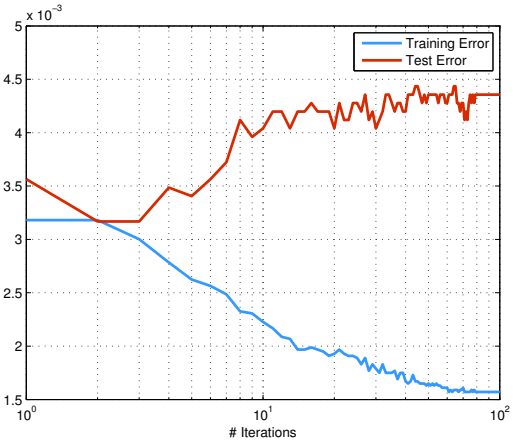


(f) GentleBoost

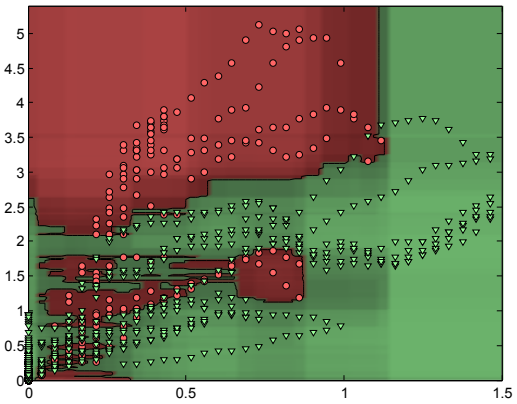
Figure B.6.: *Cost-insensitive algorithms on crash data.*



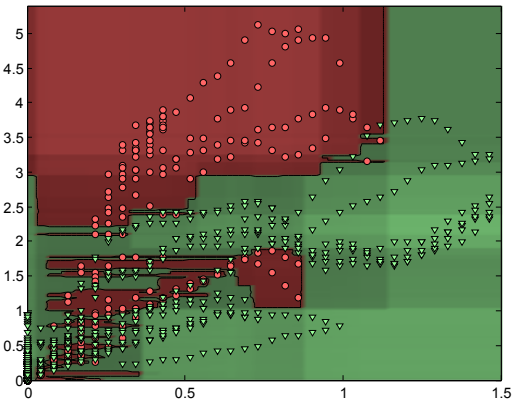
(a) LogitBoost



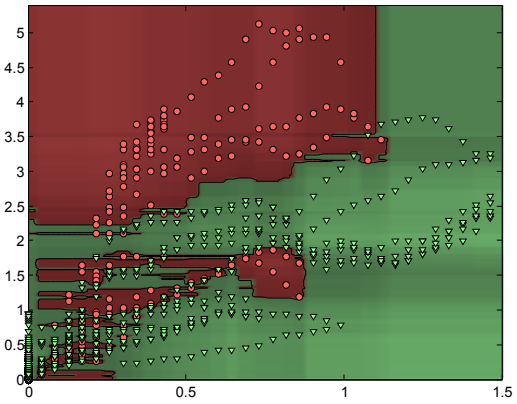
(b) LogitBoost



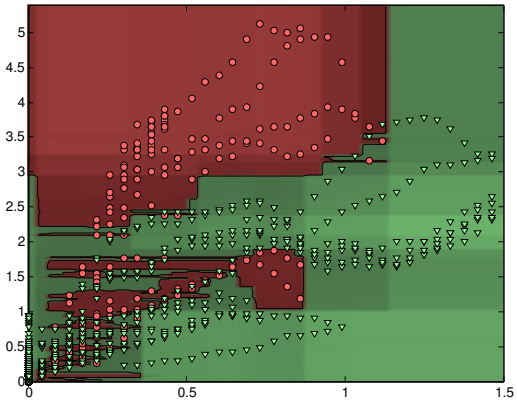
(c) AdaBoost



(d) RealBoost

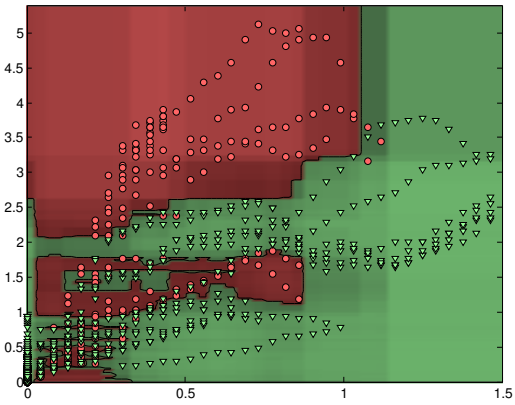


(e) GentleBoost

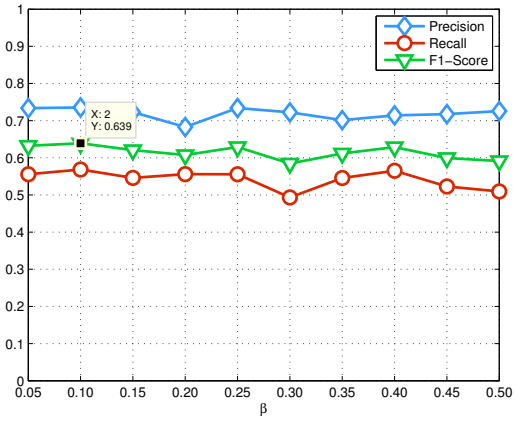


(f) LogitBoost

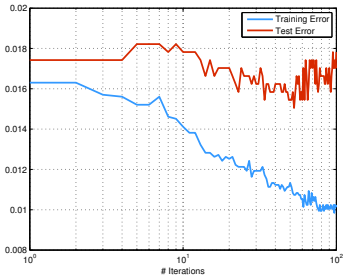
Figure B.7.: *Cost-insensitive algorithms on crash data (ctd.): Due to large areas of overlapping class distributions, there is only poor improvement of classification accuracy over 100 iterations.*



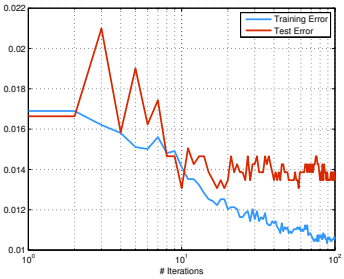
(a) Decision boundary, $\beta = 0.10$



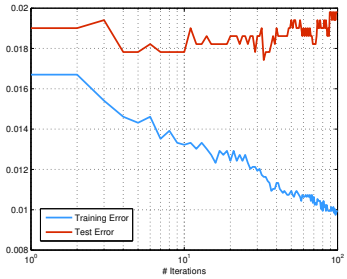
(b) PRF for different β, best at $\beta = 0.10$



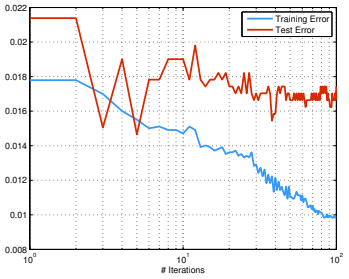
(c) $\beta = 0.05$



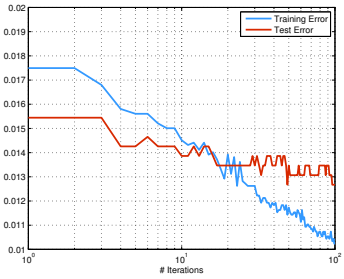
(d) $\beta = 0.10$



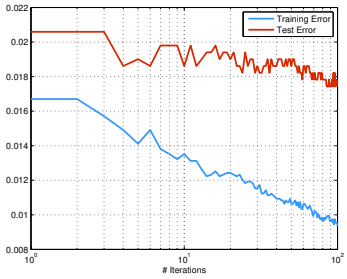
(e) $\beta = 0.15$



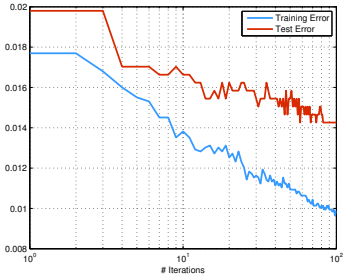
(f) $\beta = 0.20$



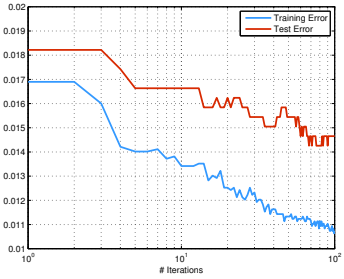
(g) $\beta = 0.25$



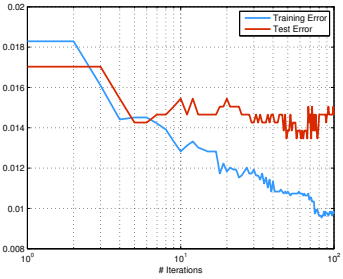
(h) $\beta = 0.30$



(i) $\beta = 0.35$

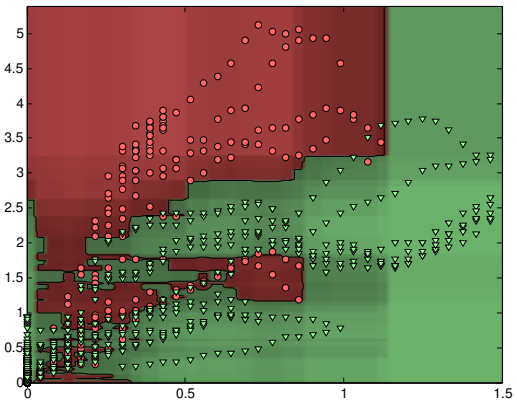


(j) $\beta = 0.40$

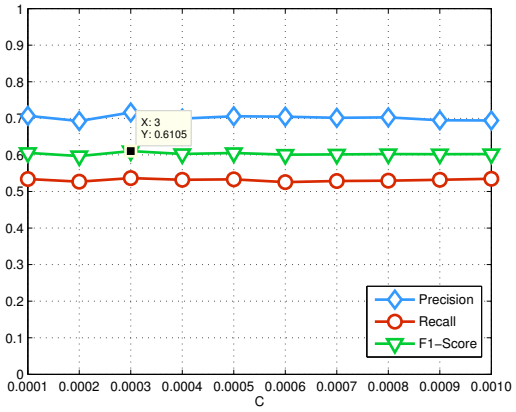


(k) $\beta = 0.45$

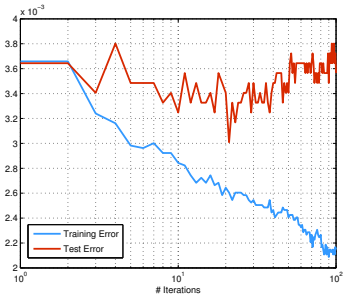
Figure B.8.: *WeightBoost on crash data.*



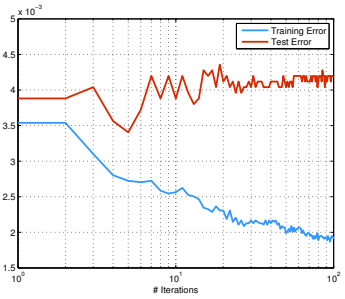
(a) Decision boundary, $C = 0.0003$



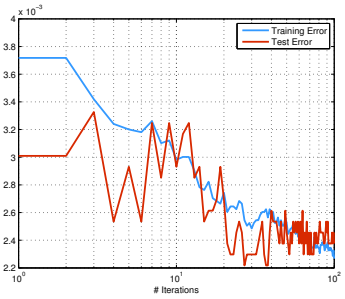
(b) PRF for different C, best at $C = 0.0003$



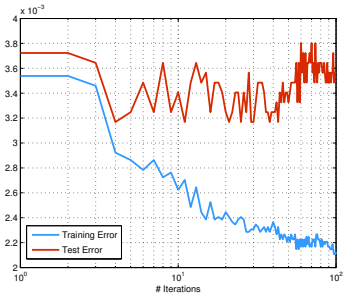
(c) $C = 0.0001$



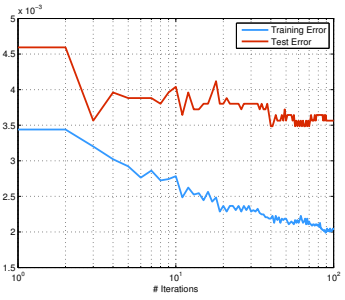
(d) $C = 0.0002$



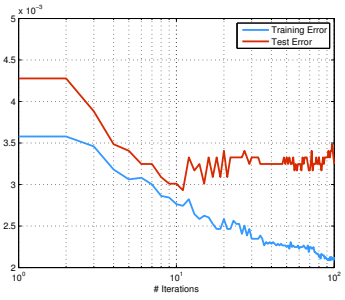
(e) $C = 0.0003$



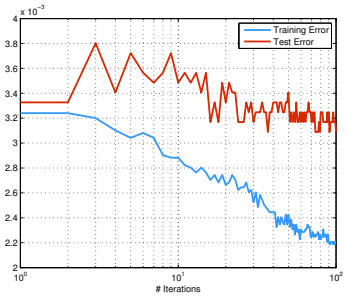
(f) $C = 0.0004$



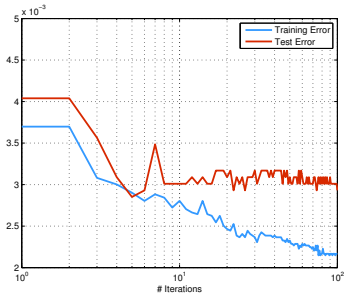
(g) $C = 0.0005$



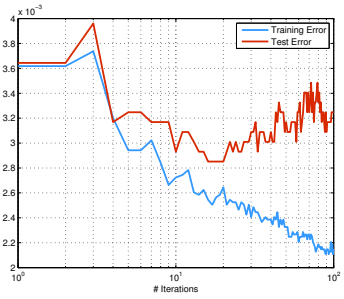
(h) $C = 0.0006$



(i) $C = 0.0007$

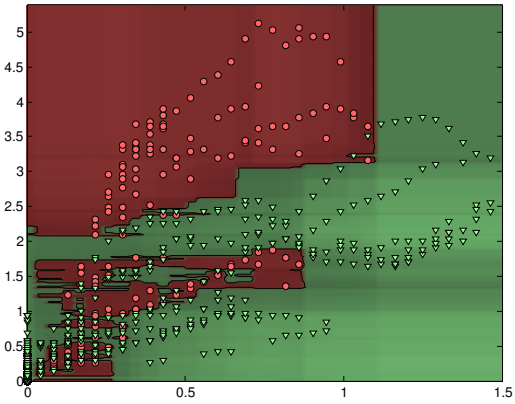


(j) $C = 0.0008$

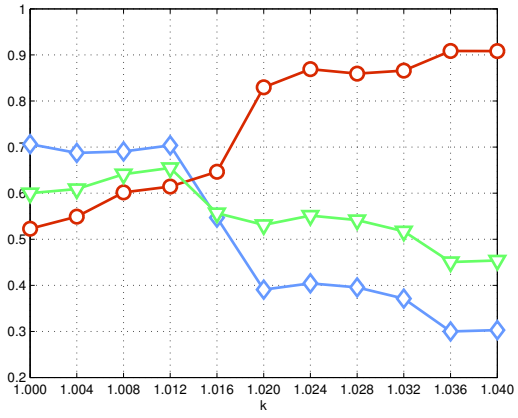


(k) $C = 0.0009$

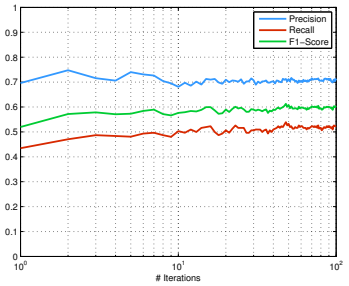
Figure B.9.: *AdaBoost$_{Reg}$ on crash data.*



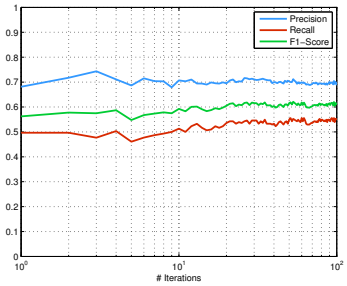
(a) Decision boundary, $k = 1.012$



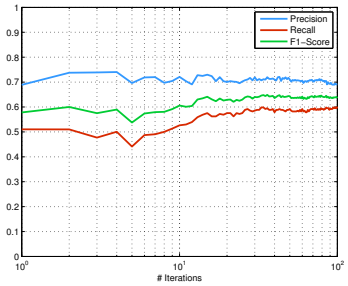
(b) PRF for different k, best at $k = 1.012$



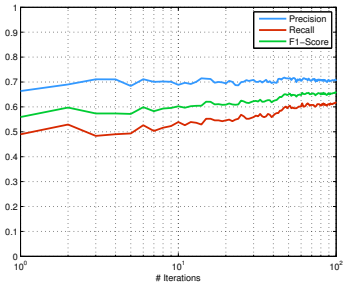
(c) $k = 1.000$



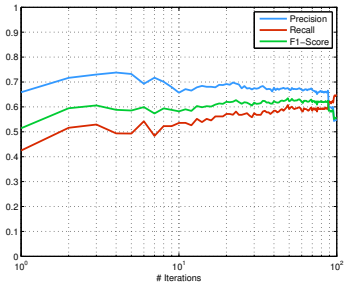
(d) $k = 1.004$



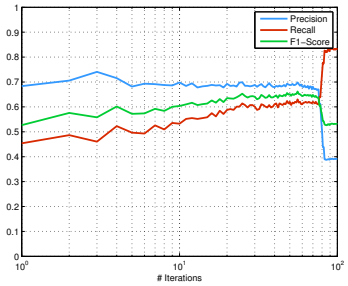
(e) $k = 1.008$



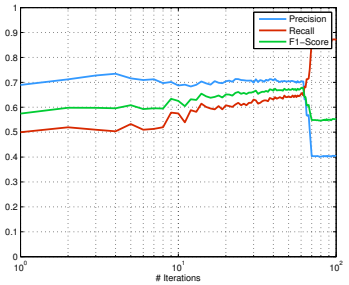
(f) $k = 1.012$



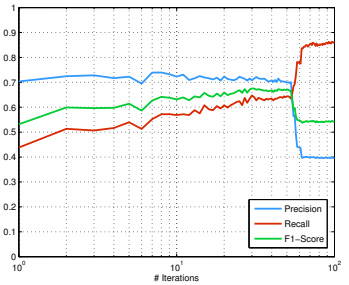
(g) $k = 1.016$



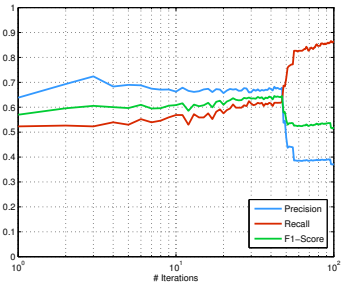
(h) $k = 1.020$



(i) $k = 1.024$

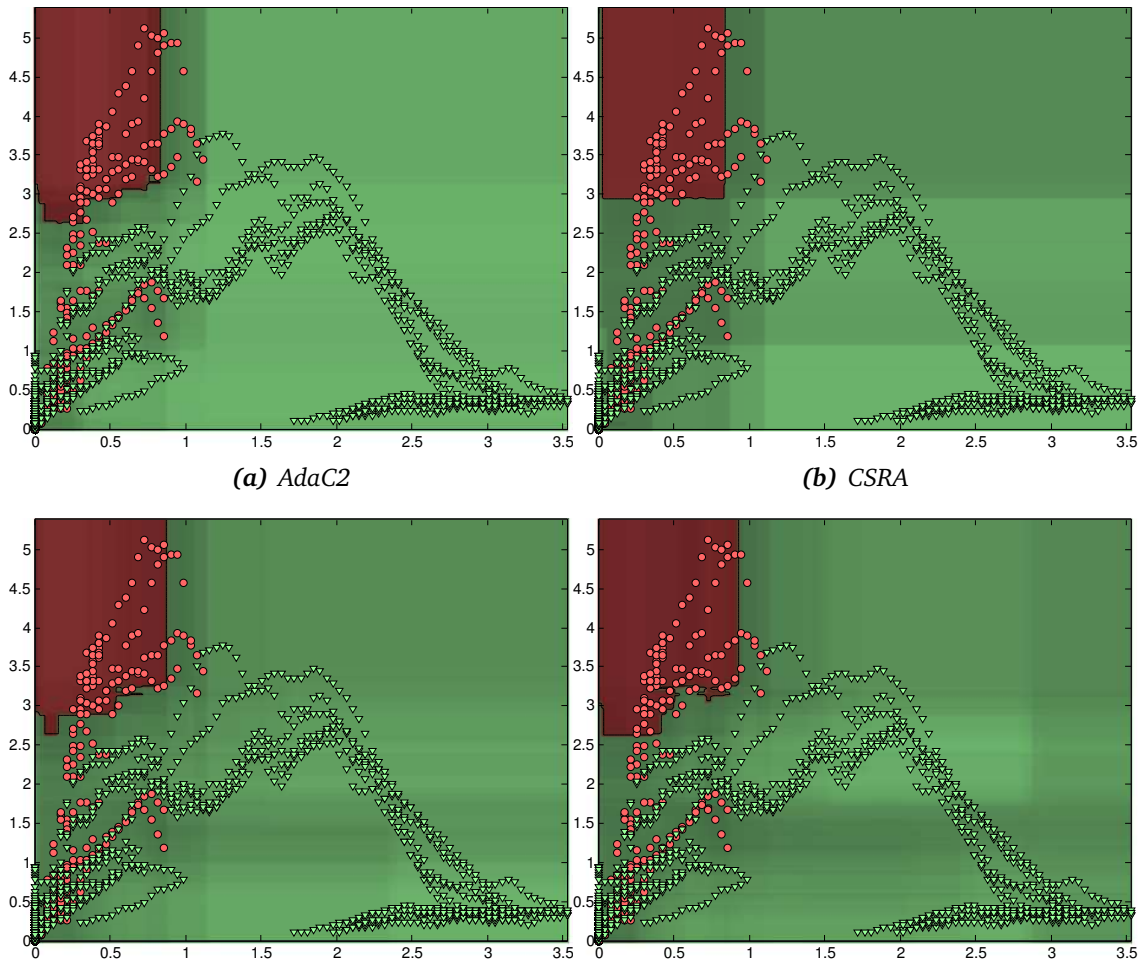


(j) $k = 1.028$



(k) $k = 1.032$

Figure B.10.: *CSGA on crash data: If costs are chosen too high, F1-score drops rapidly after a certain number of iterations.*



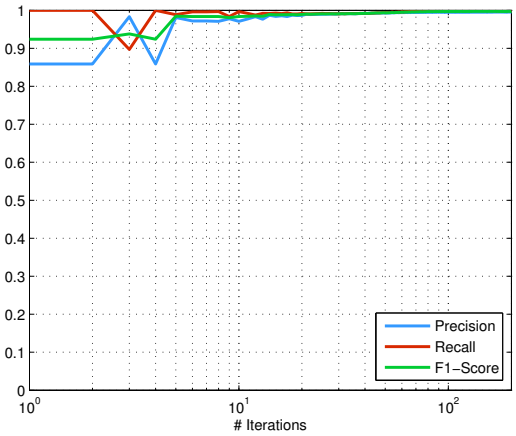
(a) *AdaC2* **(b)** *CSRA*

(c) *CSGA* **(d)** *CSLB*

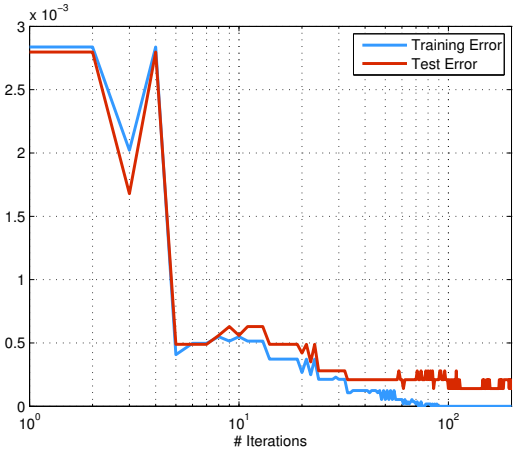
Figure B.11.: *Achieving classification goal of 100% precision on crash data by AdaC2, CSRA, CSGA and CSLB: There are no false-positive cases in the area of overlapping class distributions.*

Fire Detection Data

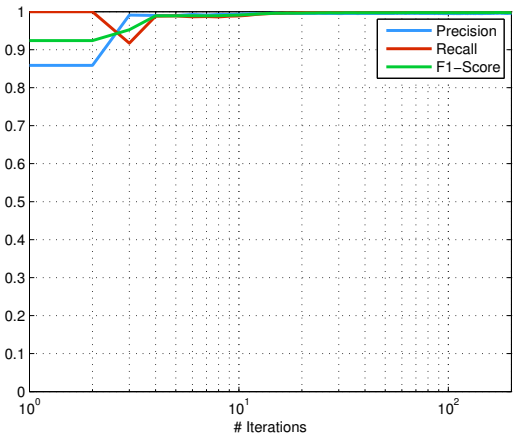
Iterations/Base Classifiers:	500 decision stumps
Evaluation Mode:	5-fold cross validation, averaged over 10 runs.



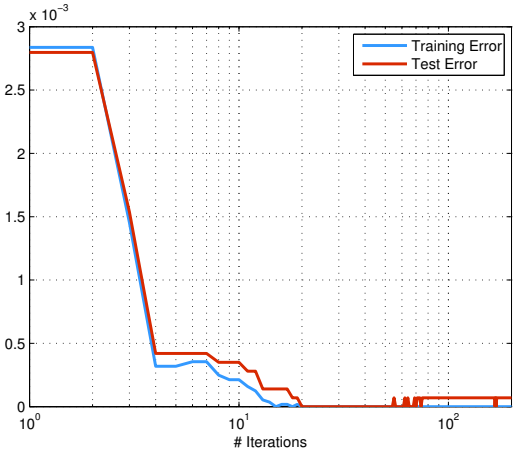
(a) AdaBoost



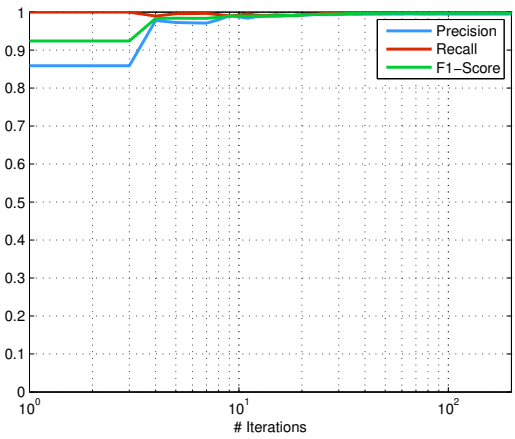
(b) AdaBoost



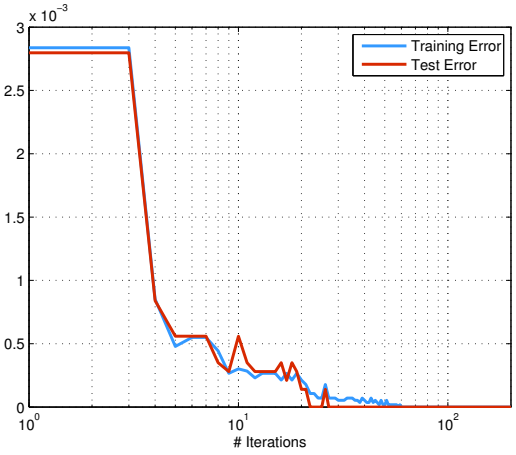
(c) RealBoost



(d) RealBoost

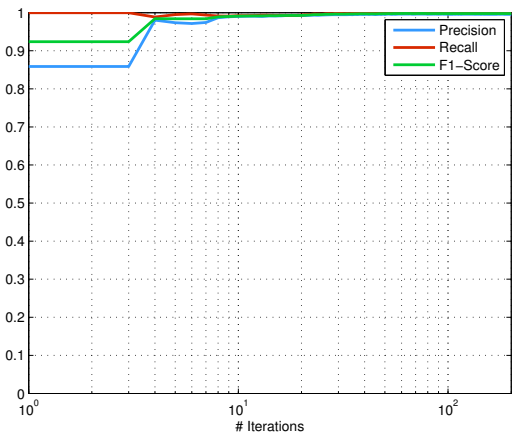


(e) GentleBoost

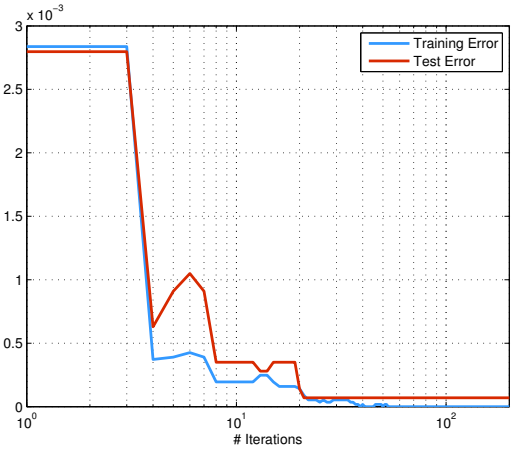


(f) GentleBoost

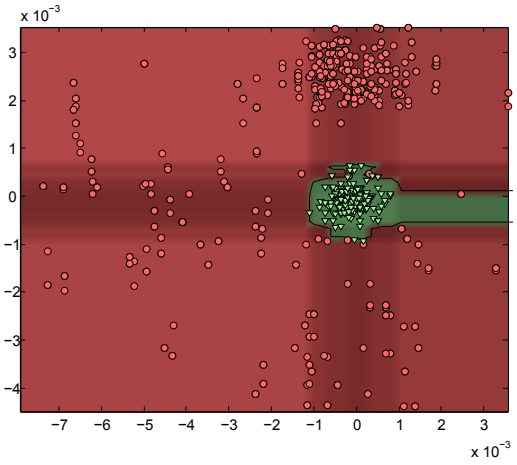
Figure B.12.: *Cost-insensitive algorithms on fire data.*



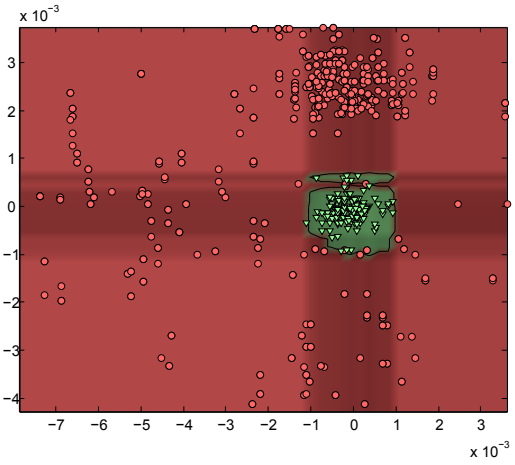
(a) LogitBoost



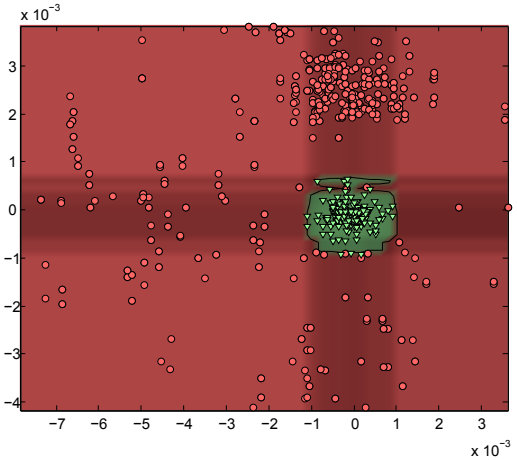
(b) LogitBoost



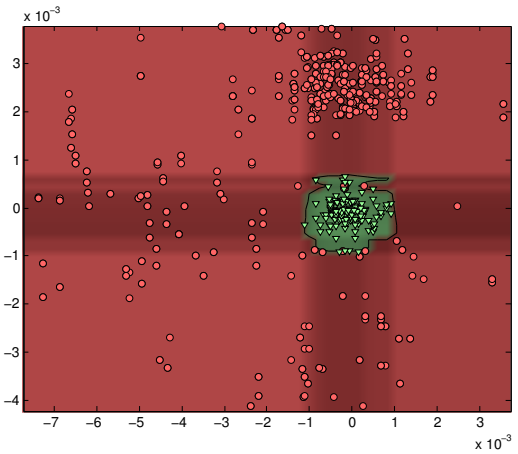
(c) AdaBoost



(d) RealBoost

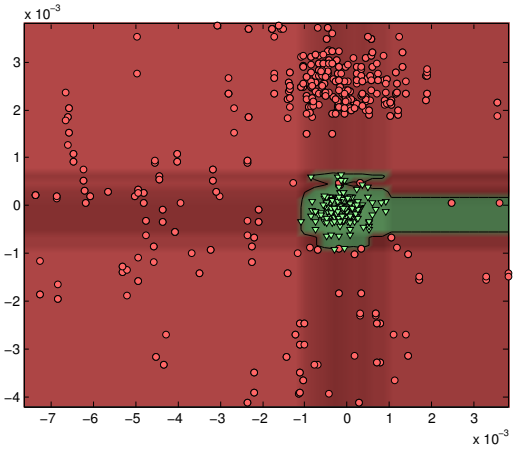


(e) GentleBoost

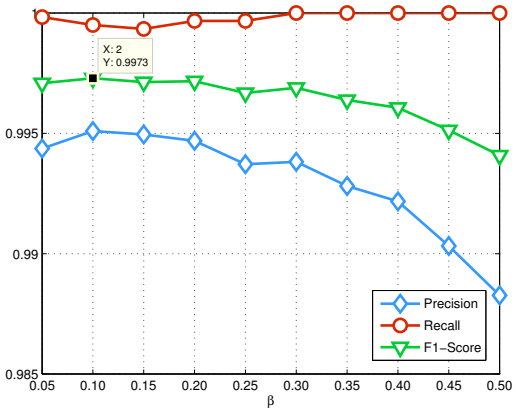


(f) LogitBoost

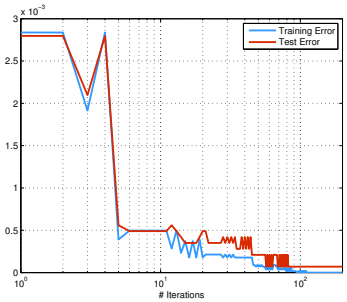
Figure B.13.: *Cost-insensitive algorithms on fire data (ctd.).*



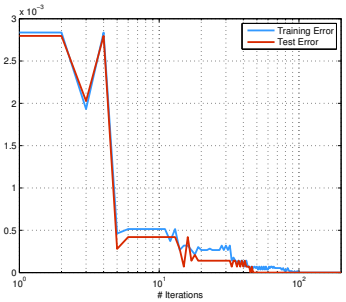
(a) Decision boundary, $\beta = 0.10$



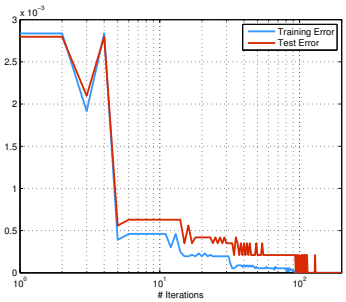
(b) PRF for different β, best at $\beta = 0.10$



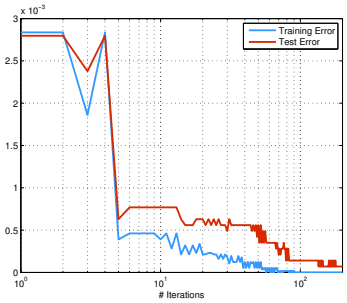
(c) $\beta = 0.05$



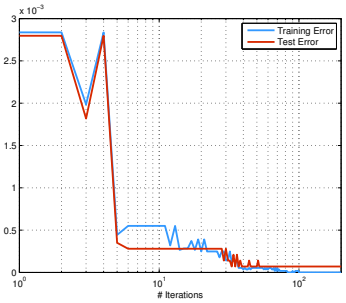
(d) $\beta = 0.10$



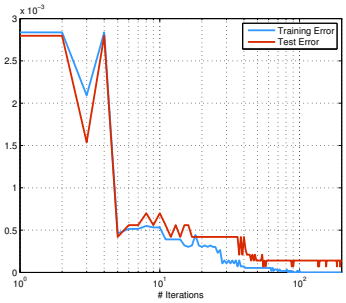
(e) $\beta = 0.15$



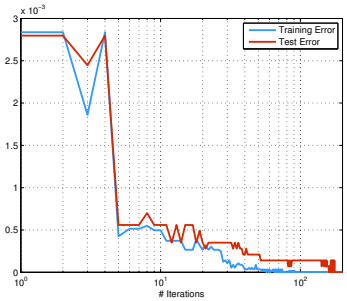
(f) $\beta = 0.20$



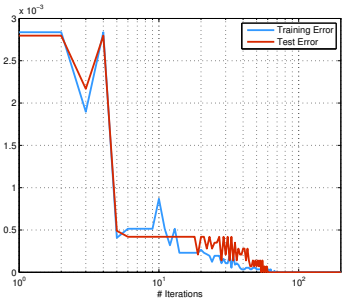
(g) $\beta = 0.25$



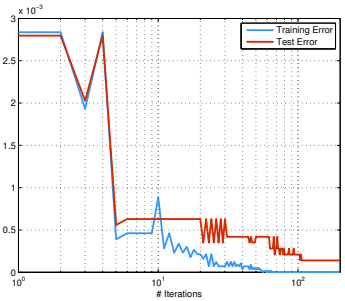
(h) $\beta = 0.30$



(i) $\beta = 0.35$

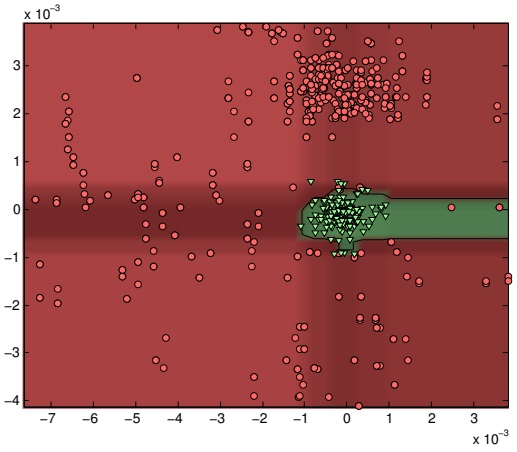


(j) $\beta = 0.40$

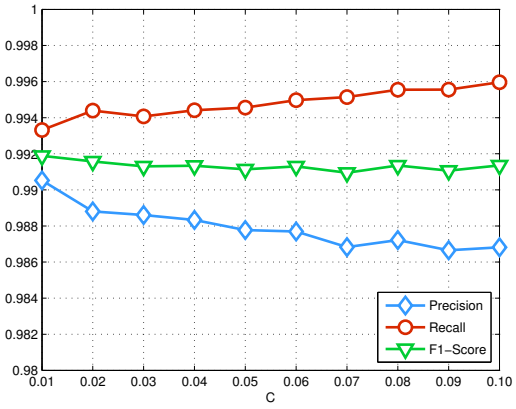


(k) $\beta = 0.45$

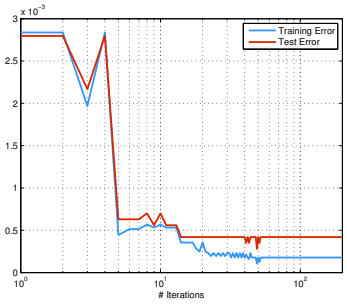
Figure B.14.: *WeightBoost on fire data.*



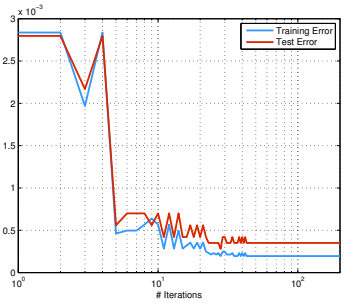
(a) *Decision boundary,* $C = 0.01$



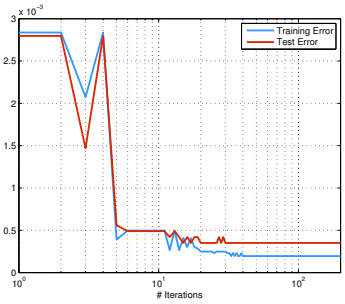
(b) *PRF for different C, best at* $C = 0.01$



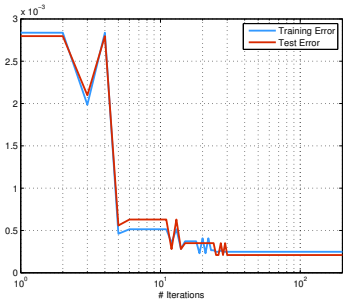
(c) $C = 0.01$



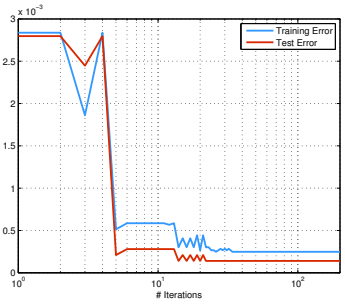
(d) $C = 0.02$



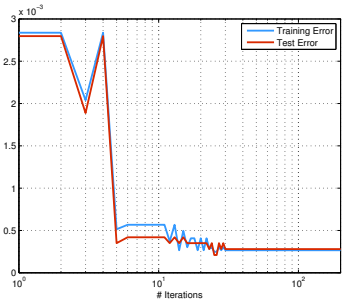
(e) $C = 0.03$



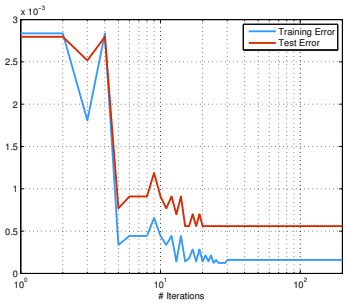
(f) $C = 0.04$



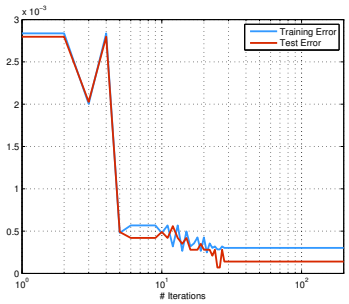
(g) $C = 0.05$



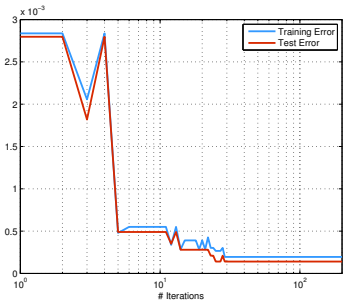
(h) $C = 0.06$



(i) $C = 0.07$

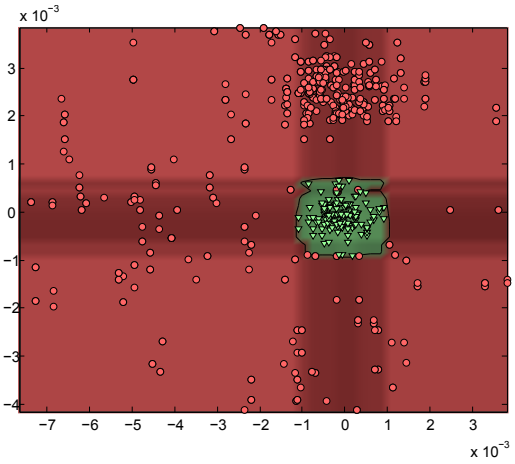


(j) $C = 0.08$

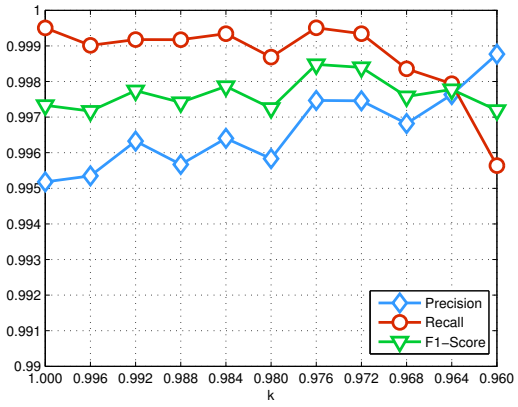


(k) $C = 0.09$

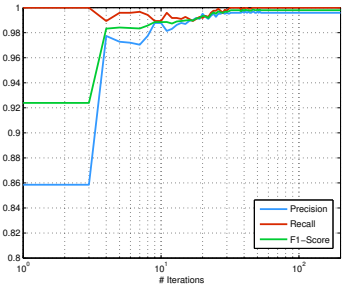
Figure B.15.: *AdaBoost*$_{Reg}$ *on fire data.*



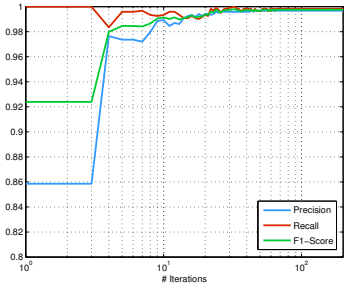
(a) *Decision boundary,* $k = 0.976$



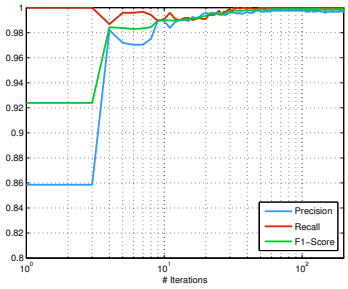
(b) *PRF for different k, best at* $k = 0.976$



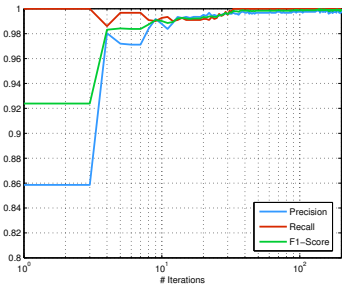
(c) $k = 1.000$



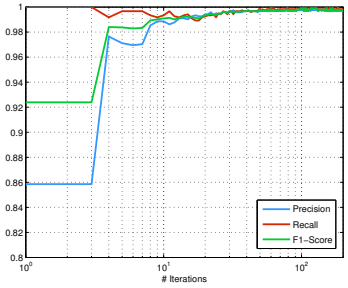
(d) $k = 0.996$



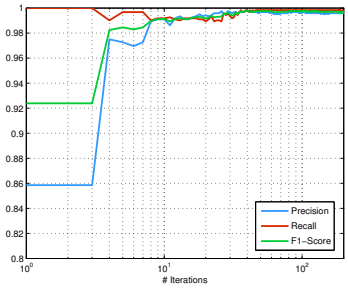
(e) $k = 0.992$



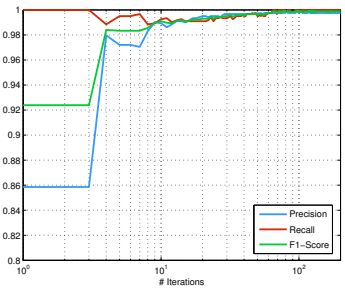
(f) $k = 0.988$



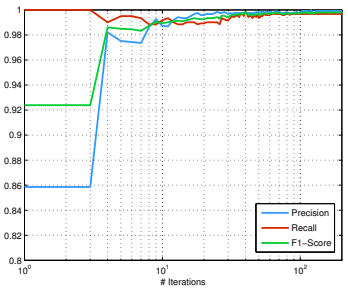
(g) $k = 0.984$



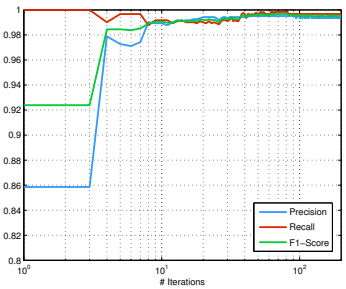
(h) $k = 0.980$



(i) $k = 0.976$



(j) $k = 0.972$

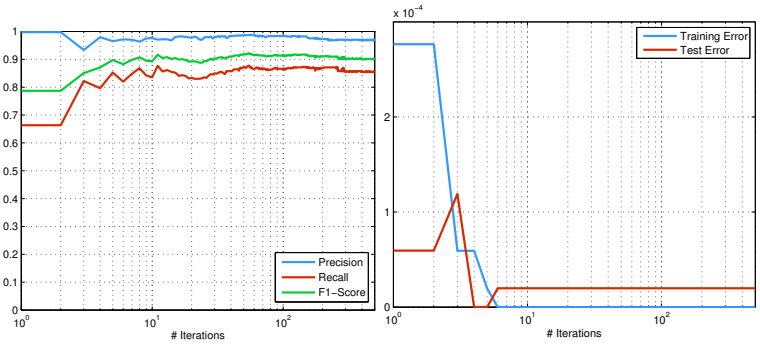


(k) $k = 0.968$

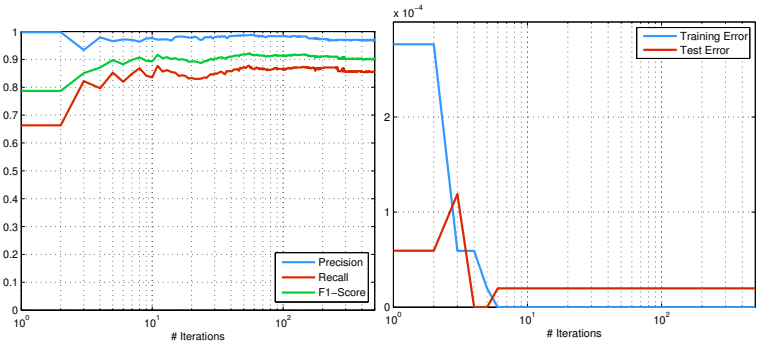
Figure B.16.: *CSGA on fire data.*

Network Intrusion Detection (IDS) Data

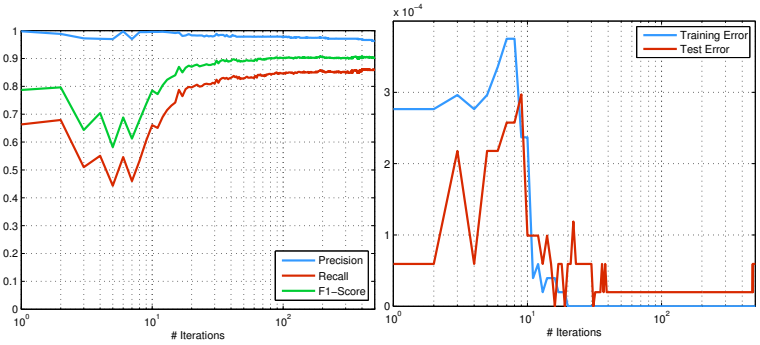
Iterations/Base Classifiers: 500 decision stumps
Evaluation Mode: Test Set Evaluation averaged over 20 Subsets.



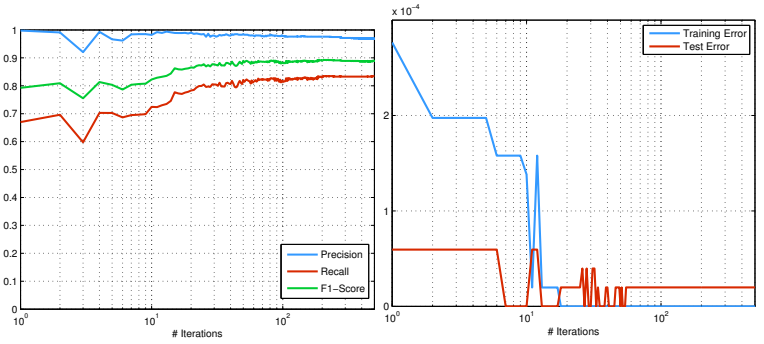
(a) *AdaBoost* **(b)** *AdaBoost*



(c) *RealBoost* **(d)** *RealBoost*

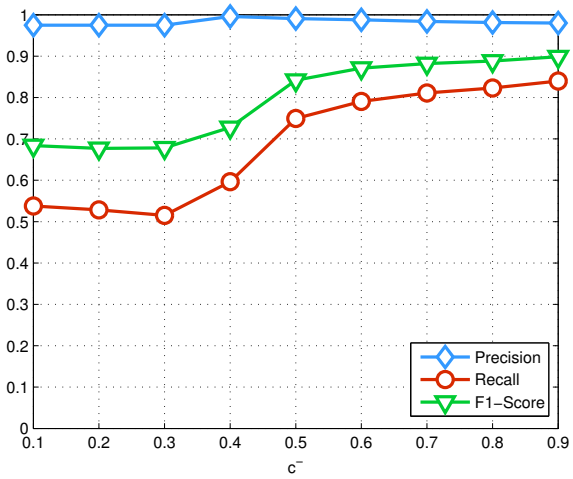


(e) *GentleBoost* **(f)** *GentleBoost*

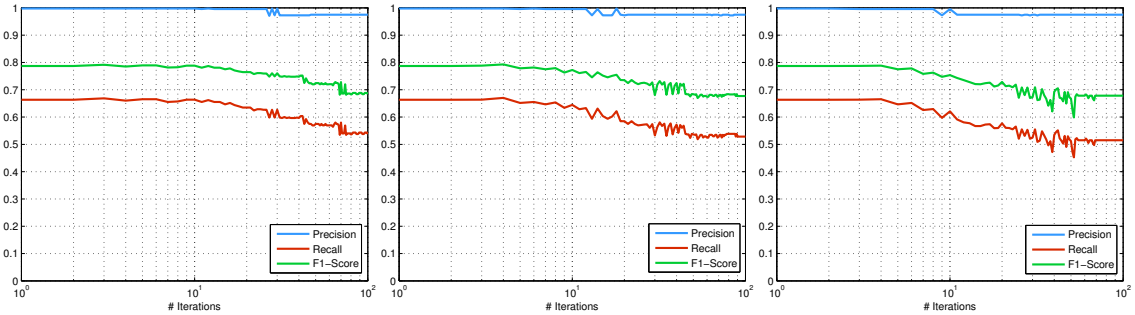


(g) *LogitBoost* **(h)** *LogitBoost*

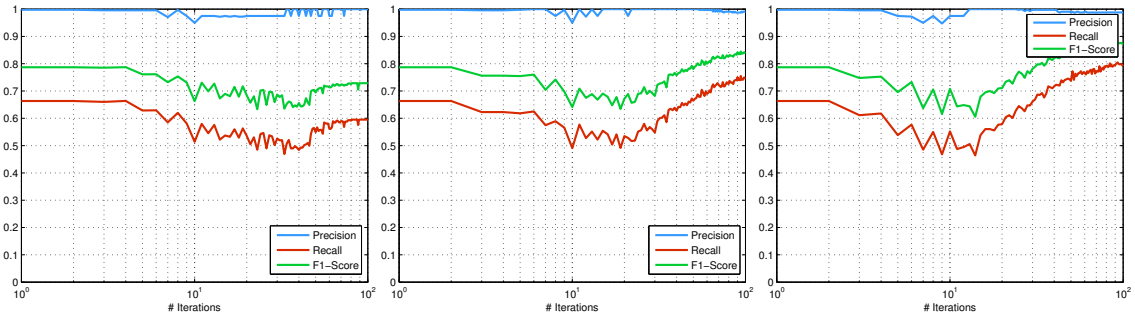
Figure B.17.: *Cost-insensitive algorithms on IDS data.*



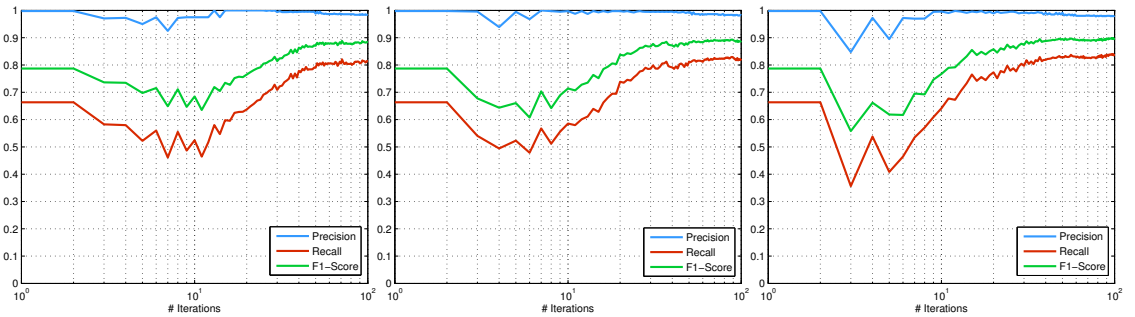
(a) PRF for different c^-, best at $c^- = 0.9$



(b) $c^+ = 1, c^- = .1$ *(c)* $c^+ = 1, c^- = .2$ *(d)* $c^+ = 1, c^- = .3$

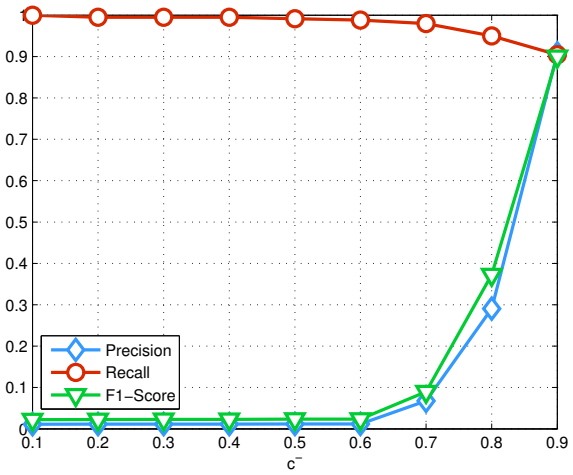


(e) $c^+ = 1, c^- = .4$ *(f)* $c^+ = 1, c^- = .5$ *(g)* $c^+ = 1, c^- = .6$

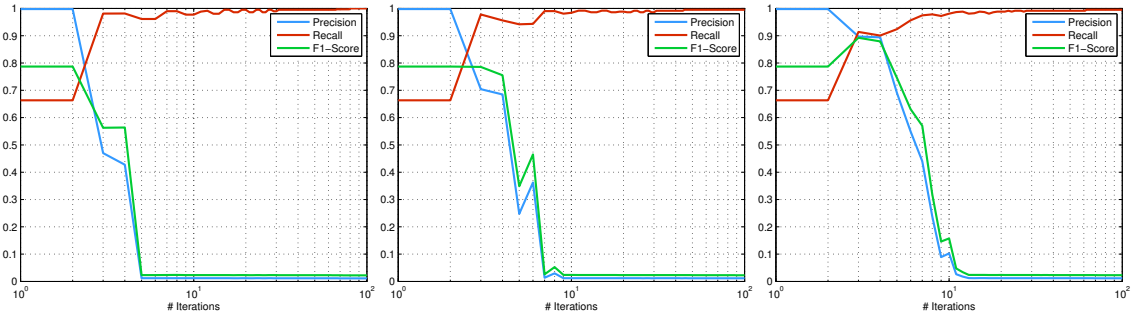


(h) $c^+ = 1, c^- = .7$ *(i)* $c^+ = 1, c^- = .8$ *(j)* $c^+ = 1, c^- = .9$

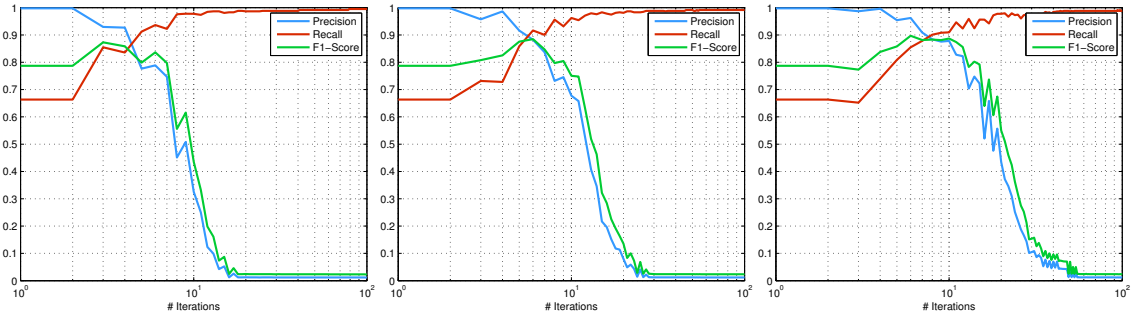
Figure B.18.: *AdaC1 on IDS data*



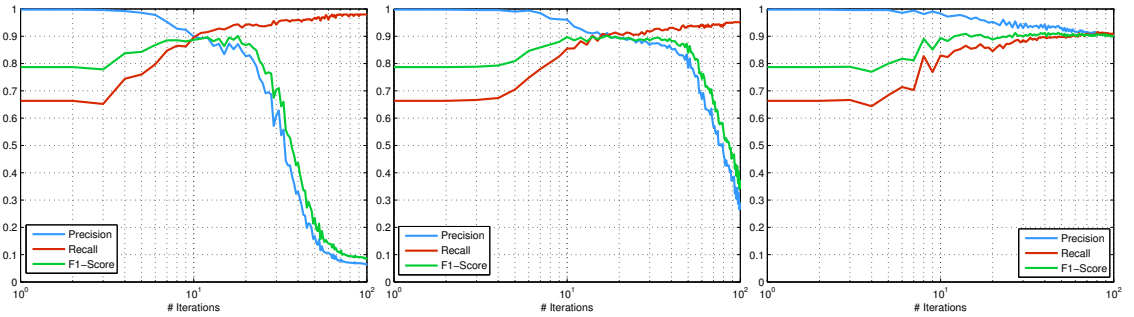
(a) *PRF for different* c^-*, best at* $c^- = 0.9$



(b) $c^+ = 1, c^- = .1$ **(c)** $c^+ = 1, c^- = .2$ **(d)** $c^+ = 1, c^- = .3$

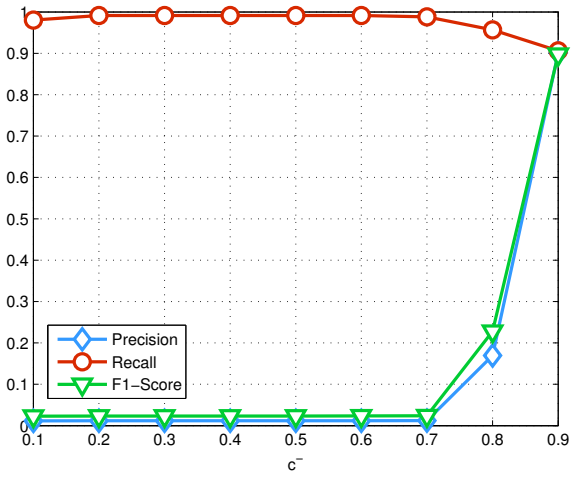


(e) $c^+ = 1, c^- = .4$ **(f)** $c^+ = 1, c^- = .5$ **(g)** $c^+ = 1, c^- = .6$

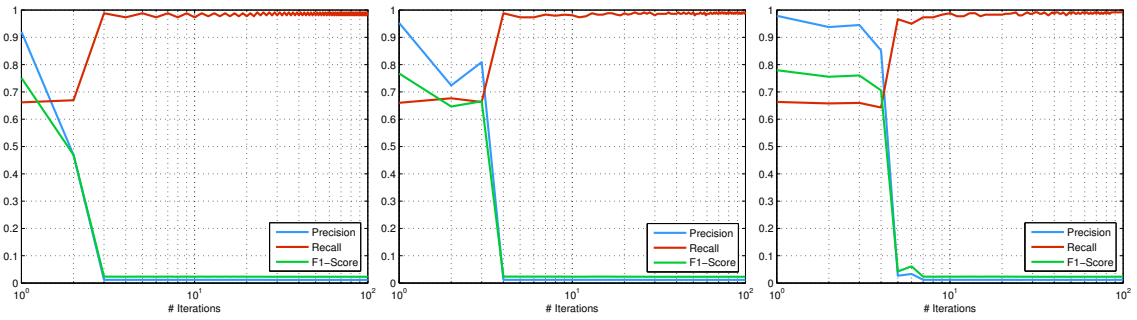


(h) $c^+ = 1, c^- = .7$ **(i)** $c^+ = 1, c^- = .8$ **(j)** $c^+ = 1, c^- = .9$

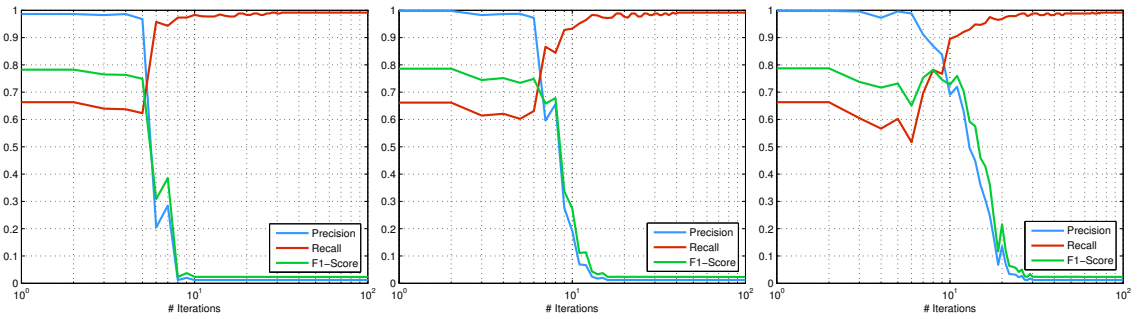
Figure B.19.: *AdaC2 on IDS data*



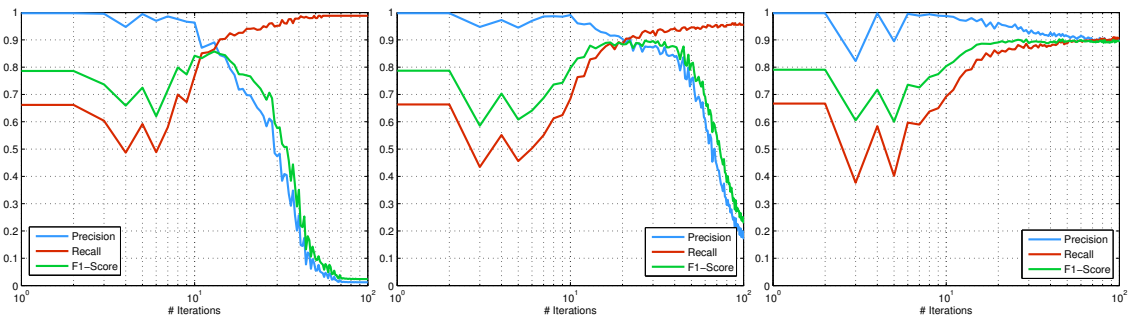
(a) *PRF for different c^-, best at $c^- = 0.9$*



(b) $c^+ = 1, c^- = .1$ **(c)** $c^+ = 1, c^- = .2$ **(d)** $c^+ = 1, c^- = .3$

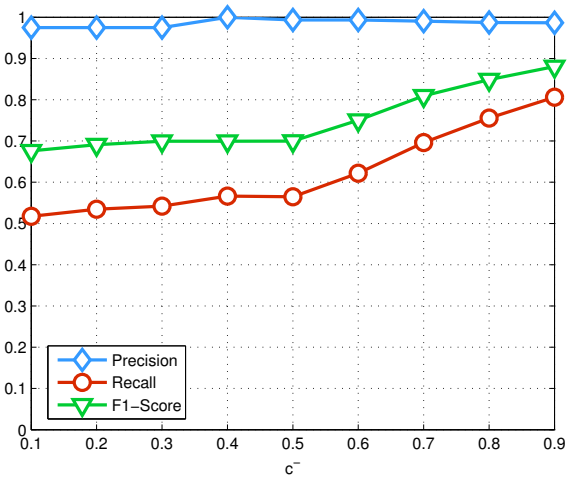


(e) $c^+ = 1, c^- = .4$ **(f)** $c^+ = 1, c^- = .5$ **(g)** $c^+ = 1, c^- = .6$

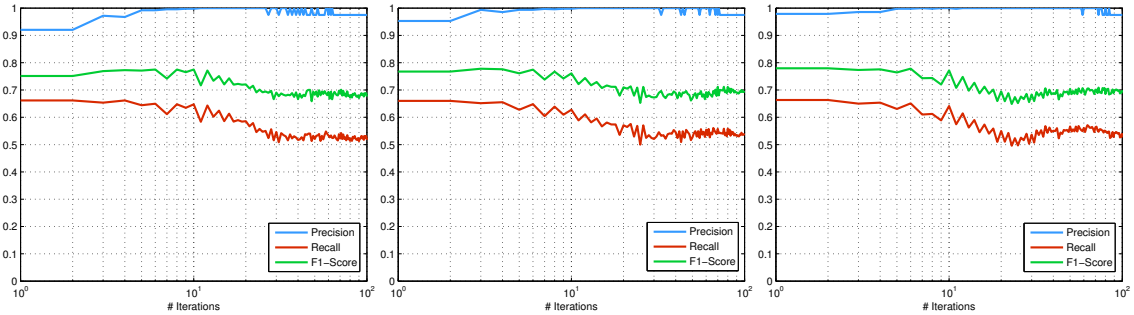


(h) $c^+ = 1, c^- = .7$ **(i)** $c^+ = 1, c^- = .8$ **(j)** $c^+ = 1, c^- = .9$

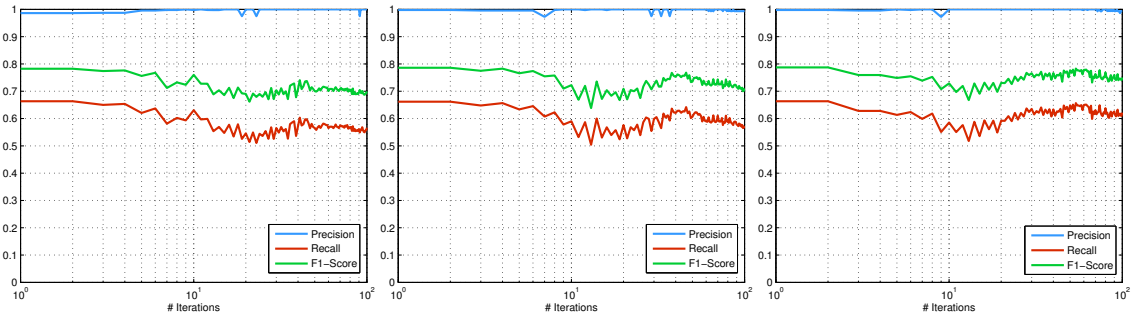
Figure B.20.: *AdaC3 on IDS data*



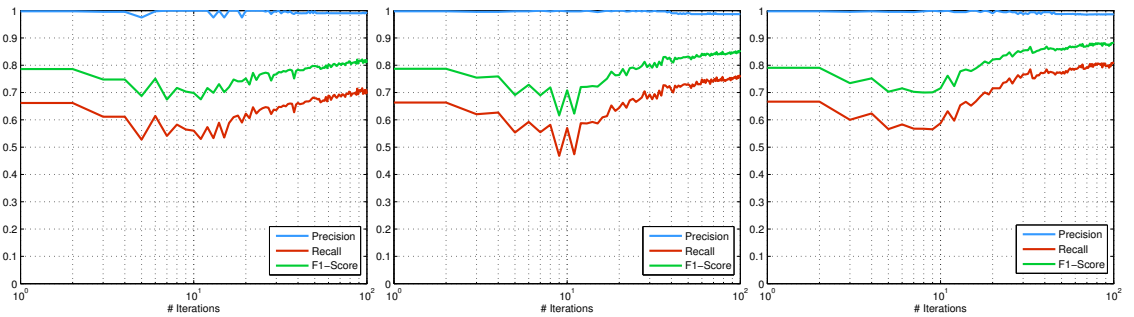
(a) *PRF for different c^-, best at $c^- = 0.9$*



(b) $c^+ = 1, c^- = .1$ *(c)* $c^+ = 1, c^- = .2$ *(d)* $c^+ = 1, c^- = .3$



(e) $c^+ = 1, c^- = .4$ *(f)* $c^+ = 1, c^- = .5$ *(g)* $c^+ = 1, c^- = .6$

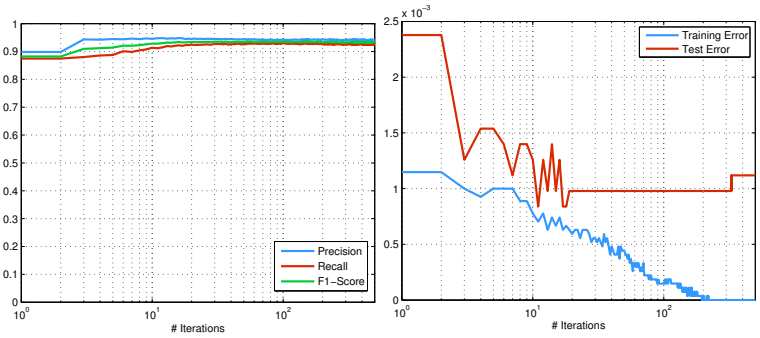


(h) $c^+ = 1, c^- = .7$ *(i)* $c^+ = 1, c^- = .8$ *(j)* $c^+ = 1, c^- = .9$

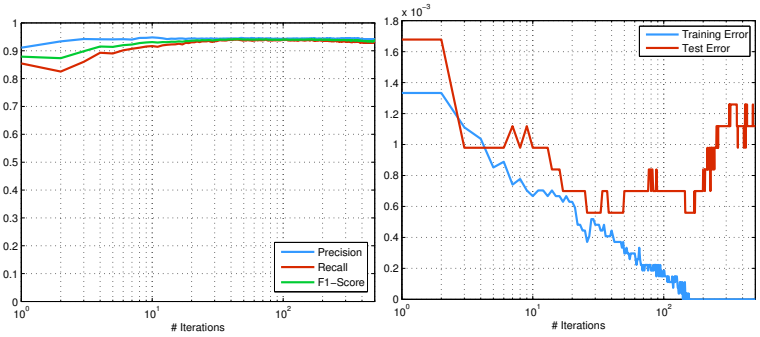
Figure B.21.: *AdaCost on IDS data*

Wisconsin Breast Cancer Data

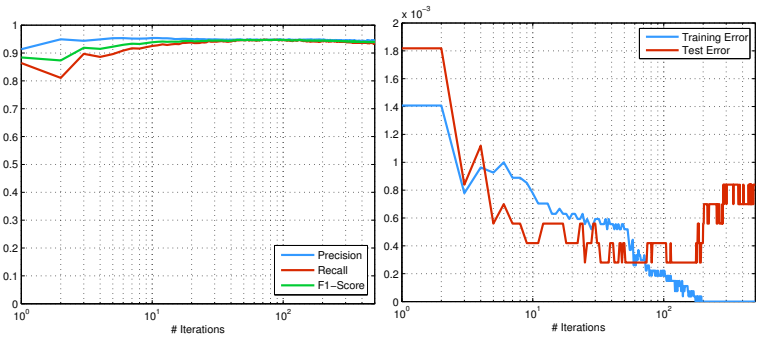
Iterations/Base Classifiers: 500 decision stumps
Evaluation Mode: 5-fold cross-validation averaged over 10 runs.



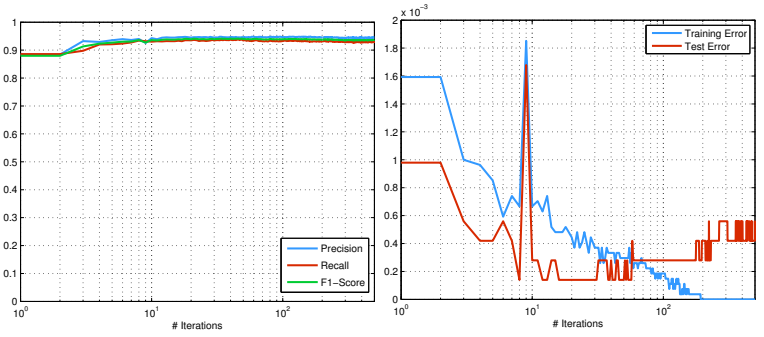
(a) *AdaBoost* **(b)** *AdaBoost*



(c) *RealBoost* **(d)** *RealBoost*

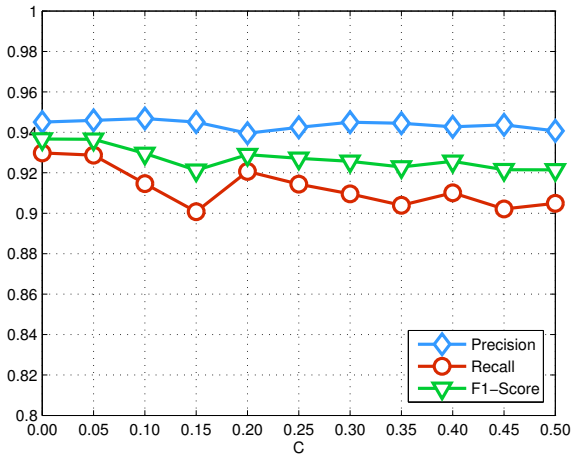


(e) *GentleBoost* **(f)** *GentleBoost*

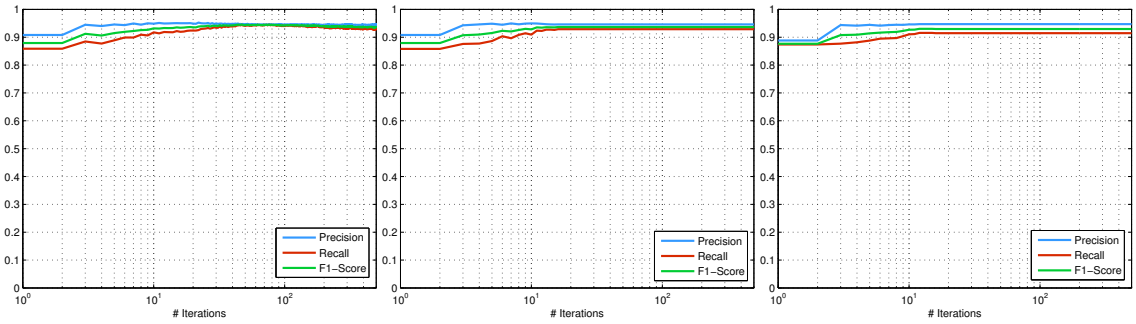


(g) *LogitBoost* **(h)** *LogitBoost*

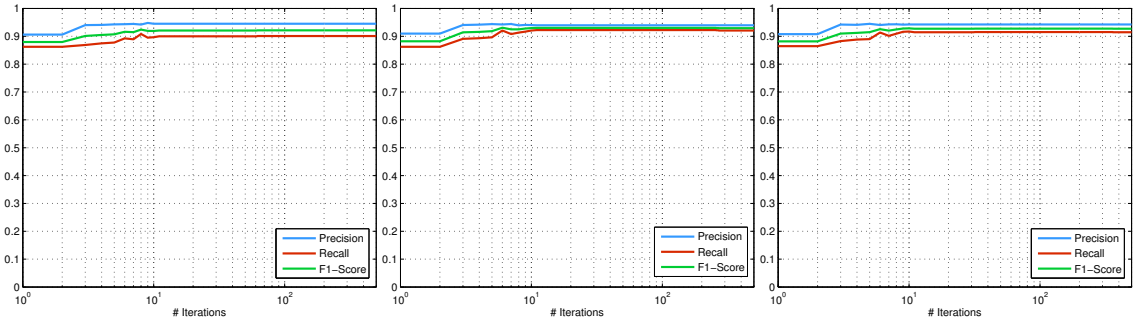
Figure B.22.: *Cost-insensitive Boosting algorithms on cancer data.*



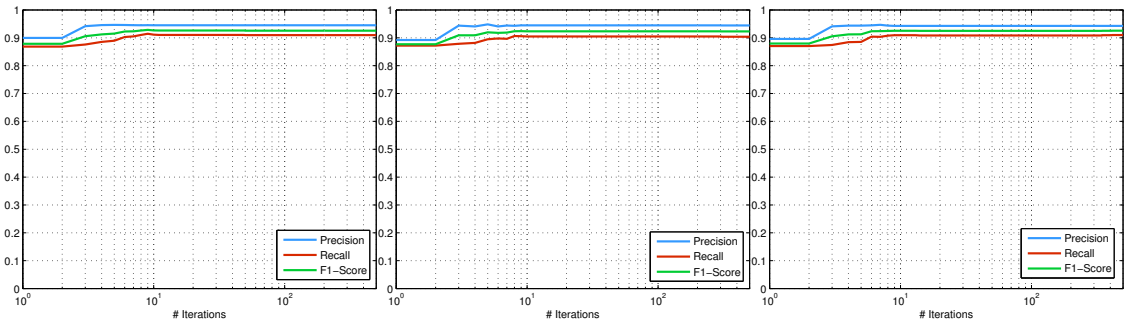
(a) *PRF for different C, best at* $C = 0.00$



(b) $C = 0.00$ **(c)** $C = 0.05$ **(d)** $C = 0.10$

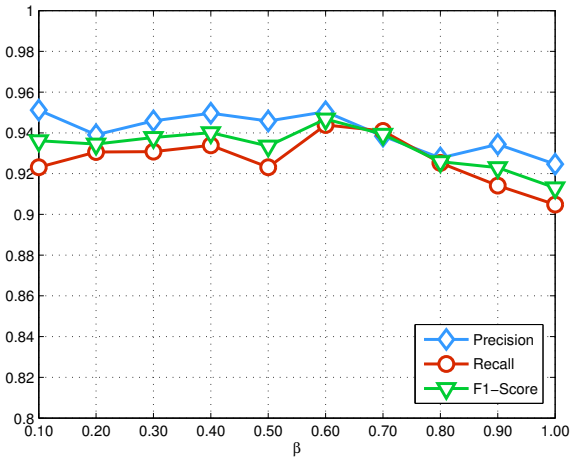


(e) $C = 0.15$ **(f)** $C = 0.20$ **(g)** $C = 0.25$

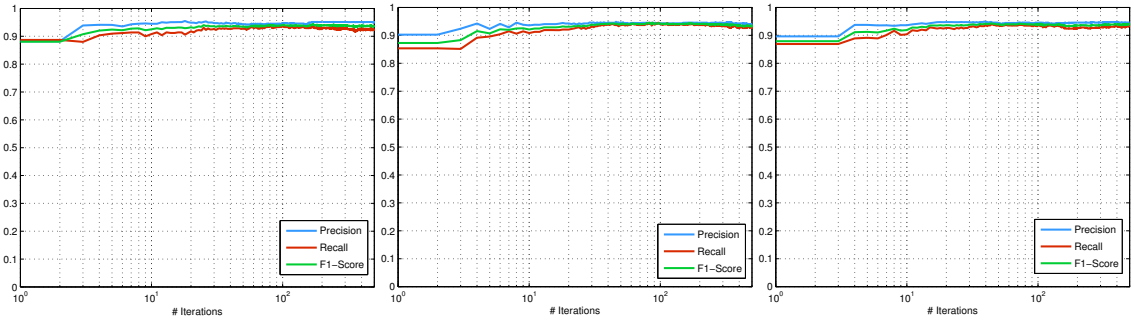


(h) $C = 0.30$ **(i)** $C = 0.35$ **(j)** $C = 0.40$

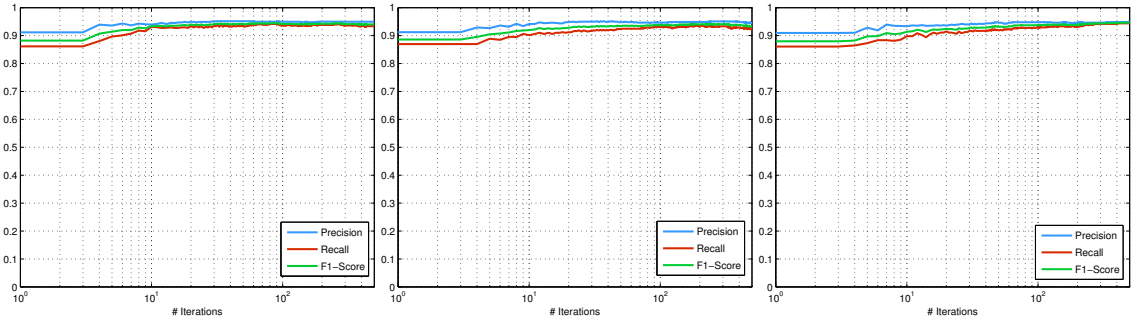
Figure B.23.: *AdaBoost*$_{Reg}$ *on cancer data.*



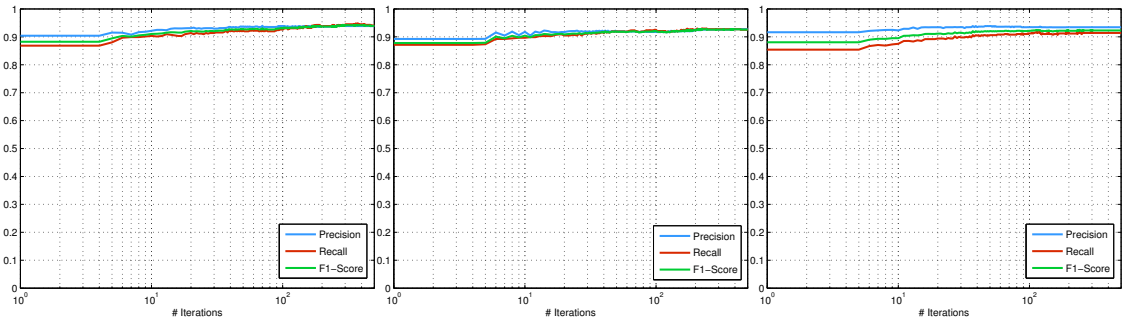
(a) PRF for different β, best at $\beta = 0.60$



(b) $\beta = 0.10$ *(c) $\beta = 0.20$* *(d) $\beta = 0.30$*

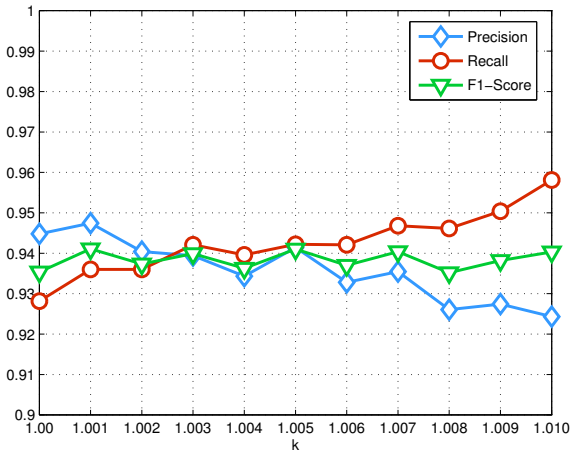


(e) $\beta = 0.40$ *(f) $\beta = 0.50$* *(g) $\beta = 0.60$*

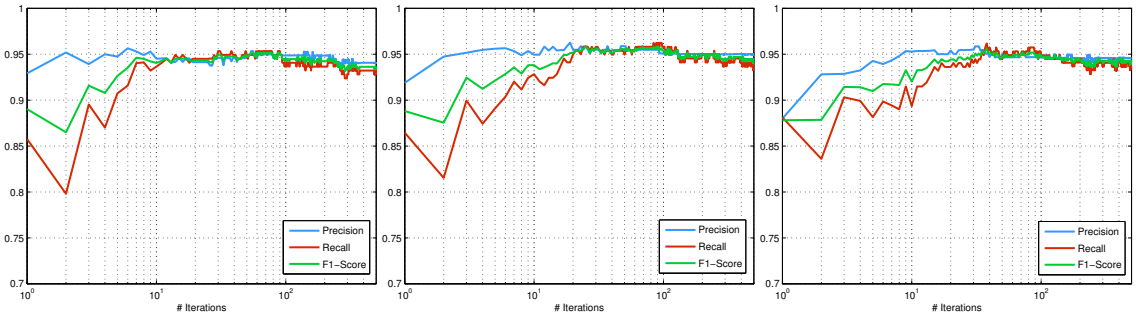


(h) $\beta = 0.70$ *(i) $\beta = 0.80$* *(j) $\beta = 0.90$*

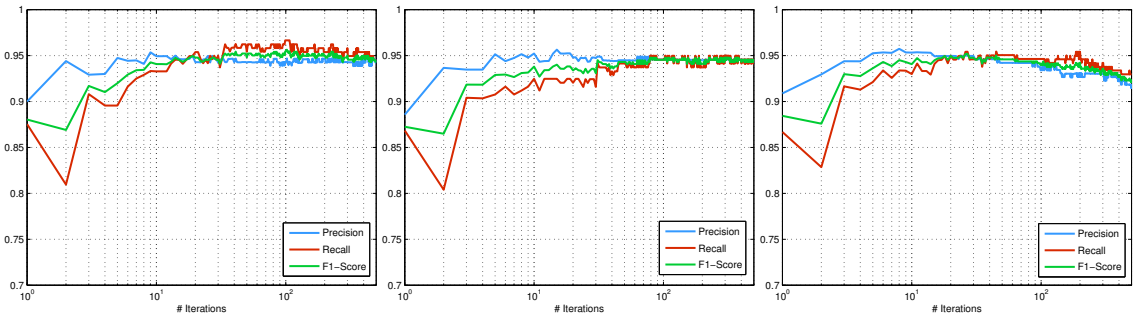
Figure B.24.: *WeightBoost on cancer data.*



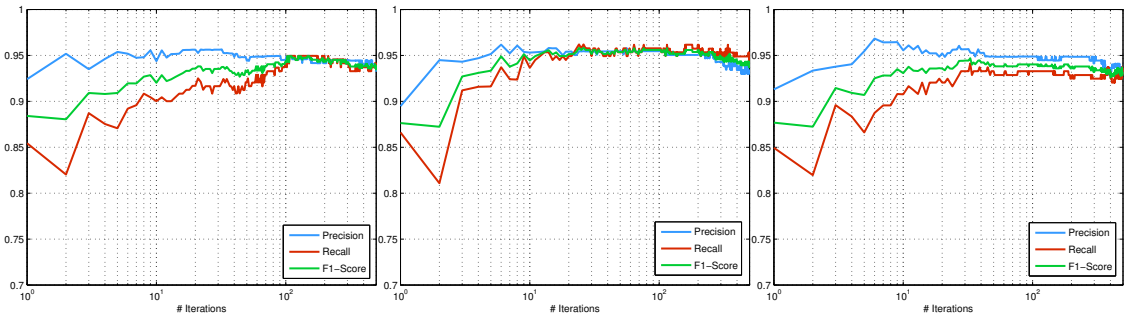
(a) *PRF for different k, best at* $k = 1.001$



(b) $k = 1.000$ **(c)** $k = 1.001$ **(d)** $k = 1.002$



(e) $k = 1.003$ **(f)** $k = 1.004$ **(g)** $k = 1.005$



(h) $k = 1.006$ **(i)** $k = 1.007$ **(j)** $k = 1.008$

Figure B.25.: *CSGA on cancer data.*

Bibliography

[1] Japkowicz, N. (editor), *Proceedings of the AAAI'2000 Workshop on Learning from Imbalanced Data Sets*, AAAI Tech Report, WS-00-05, AAAI, 2000.

[2] Chawla, N.V. and Japkowicz, N. and Kolcz, A. (editors), *Proceedings of the ICML'2003 Workshop on Learning from Imbalanced Data Sets*, 2003.

[3] Chawla, N. and Japkowicz, N. and Kolcz, A. (editors), *SIGKDD Explorations, Special Issue on Class Imbalances*, SIGKDD Explorations, 6(1), 2004.

[4] Dietterich, T. and Margineantu, D. and Provost, F. and Turney, P., editors, *Proceedings of the ICML'2000 Workshop on Cost-sensitive Learning*, 2000.

[5] He, H. and Garcia, E. A.: Learning from Imbalanced Data, *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, 2009.

[6] Freund, Y. and Schapire, R.E.: Experiments with a new Boosting Algorithm, *Proceedings of the Thirteenth International Conference on Machine Learning*, pp. 148-156, 1996.

[7] Freund, Y. and Schapire, R.E.: A Decision-Theoretic Generalization of On-Line Learning and Application to Boosting, *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119-139, 1997.

[8] Schapire, R.E.: The boosting approach to machine learning: An overview, *LECTURE NOTES IN STATISTICS*, pp. 149-172, Springer, 2003.

[9] Schapire, R. and Freund, Y.: A Short Introduction to Boosting, *Journal of Japanese Society for Artificial Intelligence*, vol. 14, no. 5, pp. 771-780, 1999.

[10] Schapire, R. and Freund, Y. and Bartlett, P. and Lee, W.S.: Boosting the Margin: A new Explanation for the Effectiveness of Voting Methods, *The Annals of Statistics*, vol. 26, no. 2, pp. 1651-1686, 1998.

[11] Friedman, J. and Hastie, T. and Tibshirani, R.: Additive logistic regression: a statistical view of boosting. *The annals of statistics*, vol. 28, no. 2, pp. 337-407, 2000.

[12] Breiman, L.: Random Forests, *Machine Learning*, vol. 45, no. 1, pp. 5-32, Springer, 2001.

[13] Japkowicz, N.: Learning from imbalanced data sets: A comparison of various strategies, *Learning from imbalanced data sets: The AAAI Workshop 10-15*. Menlo Park, CA: AAAI Press, Technical Report WS-00-05, 2000.

[14] Davis, J. and Goadrich, M.: The Relationship Between Precision-Recall and ROC Curves, *Proceedings of the 23rd International Conference on Machine Learning*, pp. 233-240, 2006.

[15] Fawcett, T.: ROC Graphs: Notes and Practical Considerations for Data Mining Researchers, *Technical Report HPL-2003-4*, HP Labs, 2003.

[16] Maloof, M.A.: Learning when data sets are Imbalanced and when costs are unequal and unknown, *ICML-2003 Workshop on Learning from Imbalanced Data Sets II*, 2003.

[17] Drummond, C. and Holte, R.: C4.5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling, *Workshop on Learning from Imbalanced Data sets II*, held in conjunction with ICML'2003, 2003.

[18] Breiman, L.: Bagging Predictors, *Technical Report 421*, Department of Statistics, University of California, 1994.

[19] Chawla, N.V. and Japkowicz, N. and Kotcz, A.: Editorial: Special Issue on Learning from Imbalanced Data Sets, *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 1, pp. 1-6, 2004.

[20] Weiss, G.M.: Mining with Rarity: A unifying Framework, *SIGKDD Explorations*, vol. 6, no. 1, 2004.

[21] Sun, Y. and Kamel, M.S. and Wong, A.K.C. and Wang, Y.: Cost-sensitive boosting for classification of imbalanced data, *Pattern Recognition*, vol. 40, no. 12, pp. 3358-3378, Elsevier, 2007.

[22] Dietterich, T.: Ensemble methods in machine learning, Multiple classifier systems, pp. 1-15, Springer, 2000.

[23] Polikar, R.: Ensemble Based Systems in Decision Making, *IEEE Circuits and Systems Magazine*, vol. 6, no. 3, pp. 21-45, 2006.

[24] Kotsiantis, S. and Kanellopoulos, D. and Pintelas, P.: Handling imbalanced datasets: A review, *GESTS International Transactions on Computer Science and Engineering*, vol. 30, no. 1, pp. 25-36, 2006.

[25] Kamel, M. and Wanas, N.: Data dependence in Combining Classifiers, *Proc. 4th Int. Workshop on Multiple Classifier Systems*, in Lecture Notes in Computer Science vol. 2709, pp. 1-14, 2003.

[26] Džeroski, S. and Ženko, B.: Is combining classifiers with stacking better than selecting the best one?, *Machine Learning*, vol. 54, no. 3, pp. 255-273, Springer, 2004.

[27] Duin, R.: The combining classifier: To train or not to train?, *International Conference on Pattern Recognition*, vol. 16, pp. 765-770, 2002.

[28] Li, Q. and Mao, Y. and Wang, Z. and Xiang, W.: Cost-Sensitive Boosting: Fitting an Additive Asymmetric Logistic Regression Model, *ACML2009, Lecture Notes in Artificial Intelligence*, Springer, 2009.

[29] Guo, H. and Viktor, H.L.: Learning from imbalanced data sets with boosting and data generation: the DataBoost-IM approach. *ACM SIGKDD Explorations Newsletter - Special issue on learning from imbalanced datasets*, Volume 6, Issue 1, June 2004.

[30] Fan, W. and Stolfo, S.J. and Zhang, J. and Chan, P.K.: AdaCost: misclassification cost-sensitive boosting, *MACHINE LEARNING-INTERNATIONAL WORKSHOP*, pp. 97-105, 1999.

[31] Schapire, R.E. and Singer, Y.: Improved boosting algorithms using confidence-rated predictions, *Machine Learning Journal*, vol. 37, no. 3, pp. 297-336, Springer, 1999.

[32] Joshi, M. and Kumar, V. and Agarwal, R.: Evaluating boosting algorithms to classify rare classes: Comparison and improvements, *First IEEE International Conference on Data Mining*, pp. 257-264, 2001.

[33] Song, J. and Lu, X. and Wu, X.: An improved AdaBoost algorithm for unbalanced classification data, *Proceedings of the 6th international conference on Fuzzy systems and knowledge discovery*, August 14-16, 2009.

[34] Ting, K.M.: A comparative study of cost-sensitive boosting algorithms, *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 983-990, 2000.

[35] Chawla, N.V. and Lazarevic, A. and Hall, L.O. and Bowyer, K.W.: SMOTEBoost: Improving prediction of the minority class in boosting, *Knowledge Discovery in Databases*, pp. 107-119, Springer, 2003.

[36] Ting, K.M. and Zheng, Z.: Boosting Trees for Cost-sensitive Classifications, *Proceedings of the First International Conference on Discovery Science*, pp. 244-255, Springer, 1998.

[37] Rätsch, G. and Onoda, T. and Müller, K.R.: Soft margins for AdaBoost, *Machine Learning*, vol. 42, no. 3, pp. 287-320, Springer, 2001.

[38] Liu, Y. and Si, L. and Carbonell, J..: A new boosting algorithm using input-dependent regularizer, *The Twentieth Conference on Machine Learning (ICML'03)*, 2003.

[39] Masnadi-Shirazi, H. and Vasconcelos, N.: Asymmetric Boosting, *Proceedings of the 24th international conference on Machine learning*, ACM, pp. 619, 2007.

[40] Hastie, T. and Tibshirani, R. and Friedman, J.: The Elements of Statistical Learning - Data Mining, Inference, and Prediction, *Springer Series in Statistics*, 2nd Edition, Springer, 2009.

[41] Bishop, C.: Pattern Recognition and Machine Learning, Springer, 2006.

[42] Kuncheva, L. I.: Combining Pattern Classifiers - Methods and Agorithms, *John Wiley & Sons, Inc.*, 2004.

[43] Lewis, D. and Gale, W.: Training Test Classifiers by Uncertainty Sampling, *Proceedings of the Seventeeth Annual International ACM SIGIR Conference on Research and Development in Informaion*, pp. 73-79, 1998.

[44] Tan, P. and Steinbach, M. and Kumar, V.: Introduction to Data Mining, *Addison-Wesley*, 2006.

[45] Hanson, L. K. and Salamon, P.: Neural Network Ensembles, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp.993-1991, 1990.

[46] Efron, B. and Tibshirani, R.: An Introduction to the Boostrap, *Chapman & Hall, NY*, 1993.

[47] Breiman, L.: Classification and Regression Trees, *Chapman & Hall*, 1984.

[48] Domingos, P.: Why does Bagging work? A Bayesian Account and its implications, *Proceedings of the third International Conference on Knowledge Discovery and Data Mining*, pp. 155-158, 1997.

[49] Breiman, L.: Pasting Small Votes for Classification in Large Databases and On-Line, *Machine Learning*, vol. 36, no. 1, pp. 85-103, 1999.

[50] Rudin, C. and Schapire, R.E. and Daubechies, I.: Boosting Based on a Smooth Margin, *Learning Theory*, pp. 502-517, Springer, 2004.

[51] Cortes, C. and Vapnik, V.: Support Vector Networks, *Machine Learning*, vol. 20, pp. 273-297, 1995.

[52] Boser, B. and Guyon, I. and Vapnik V.: A Training Algorithm for Optimal Margin Classifiers, *Fifth Annual ACM Workshop on COLT, pp. 144-152, ACM Press, 1992.*

[53] Weiss, G. and Provost, F.: The effect of class distribution on classifier learning: an empirical study, *Rutgers Univ*, 2001.

[54] Weiss, G. and Provost, F.: Learning when Training Data are Costly: The Effect of Class Distribution on Tree Induction, *Journal for Artificial Intelligence Research*, vol. 19, pp. 315-354, 2003.

[55] Kubat, M. and Matwin, S.: Addressing the Curse of Imbalanced Training Sets: One Sided Selection, *Proceedings of the Fourteenth International Conference on Machine Learning*, Morgan Kaufmann, pp. 179-186, 1997.

[56] Chawla, N. V. and Hall, L. O. and Bowyer, K. W. and Kegelmeyer, W. P.: SMOTE: Synthetic Minority Oversampling TEchnique, *Journal of Artificial Intelligence Research*, vol. 16, pp. 321-357, 2002.

[57] Raskutti, B. and Kowalczyk, A.: Extreme Rebalancing for SVMs: A Case Study, *SIGKDD Explorations*, vol. 6, no. 1, pp. 60-69, 2004.

[58] Schapire, R.: The Strength of Weak Learnability, *Machine Learning*, vol. 5, no. 2, pp. 197-227, 1990.

[59] Blumer, A. and Ehrenfeucht, A. and Haussler, D. and Warmuth, M.K.: Learnability and the Vapnik-Chervonenkis Dimension, *Journal of the ACM*, vol. 36, no. 4, pp. 929-965, 1989.

[60] Vapnik, V.N. and Chervonenkis, A.Y.: On Uniform Convergence of the Frequencies of Events to their Pobebility, *Theory of Probability and its Applications*, vol. 16, no. 2, pp. 264-280, 1971.

[61] Breiman, L.: Arcing Classifiers, *The Annals of Statistics*, vol. 26, no. 3, pp. 801-849, 1998.

[62] Krieger, A. and Long, C. and Wyner, A.: Boosting Noisy Data, *International Conference on Machine Learning*, pp. 274-281, 2001.

[63] Quinlan, J.: Boosting First-Order Learning, *Lecture Notes in Computer Sciences*, pp. 143-155, Springer, 1996.

[64] Grove, A.J. and Schuurmans, D.: Boosting in the Limit: Maximizing the Margin of Learned Ensembles, *Proceedings of the National Conference on Artificial Intelligence*, pp. 692-699, 1998.

[65] Drucker, H. and Cortes, C.: Boosting Decision Trees, *Advances in Neural Information Processing Systems*, vol. 8, pp. 479-485, 1996.

[66] Quinlan, J.R.: Bagging, Boosting and C4.5, *Programs for Machine Learning*, Morgan Kaufmann, 1993.

[67] Rieck, K.: Machine Learning for Application-Layer Intrusion Detection, Dissertation, 2009.

[68] Nusser, S. and Otte, C. and Hauptmann, W. and Leirich, O. and Krätschmer, M. and Kruse, R.: Machine Learning of Verifiable Classifiers for Autonomous Control of Safetly-related Systems, *at - Automatisierungstechnik*, vol. 57, no. 3, pp. 138-145, 2009.

[69] Mangasarian, O.L. and Wolberg, W.H.: Cancer diagnosis via linear programming, *SIAM News*, vol. 23, no. 5, pp. 1-18, 1990.

[70] Frank, A. and Asuncion, A.: UCI Machine Learning Repository, *http://archive.ics.uci.edu/ml*, University of California, Irvine, School of Information and Computer Sciences, 2010.

[71] Zhu, X., Wu, X.: Class Noise vs. Attribute Noise: A quantitative Study, *Artificial Intelligence Review*, Springer, 2004.

[72] Press, W. and Flannery, B. and Teukolsky, S. and Vetterling, W.: Numerical Recipes in C, *Cambridge University Press*, Second Edition, 1992.

[73] Valiant, L.G.: A Theory of the Learnable, Communications of the ACM, vol. 27, no. 11, pp. 1134-1142, 1984.

[74] Fawcett, T.: An Introduction to ROC Analysis, *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861-874, Elsevier, 2006.

[75] Sochman, J. and Matas, J.: AdaBoost, *Lecture Slides*, Center for Machine Perception, Czech Technical University, Prague, `http://cmp.felk.cvut.cz/~sochmj1/adaboost_talk.pdf`, *(accessed on 10/30/2010)*.

[76] Wang, Z. and Fang, C. and Ding, X.: Symmetric Real AdaBoost, *19th International Conference on Pattern Recognition (ICPR)*, pp. 1-4, 2009.

[77] Chang, C.C. and Chih-Jen, L.: LIBSVM, http://www.csie.ntu.edu.tw/ cjlin/libsvm/

[78] Quinlan, J.R.: Induction of Decision Trees, *Machine Learning*, vol. 1, no. 1, pp. 81-106, Springer, 1086.

[79] Sun, Y. and Kamel, M.S. and Wang, Y.: Boosting for Learning Multiple Classes with Imbalanced Class Distribution, *Sixth International Conference on Data Mining, ICDM'06*, pp. 592-602, 2006.