D.A. Vente

A practical comparison of boosting algorithms

bachelor's thesis

supervised by
dr. T.A.L. van Erven

June 19, 2016

Universiteit Leiden
Wiskunde en Natuurwetenschappen

Mathematical Institute
Leiden University

Contents

1 Introduction **1**
1.1 Outline . . . 1
1.1.1 Setting . . . 1
1.1.2 Scope . . . 1
1.1.3 Basic structure . . . 1

2 The three boosting algorithms **2**
2.1 Settings . . . 2
2.1.1 Classification . . . 2
2.1.2 Hedging . . . 2
2.1.3 Weak learners . . . 3
2.1.4 Boosting . . . 3
2.2 **AdaBoost** . . . 4
2.2.1 The algorithm . . . 4
2.2.2 The theoretical performance of **AdaBoost** . . . 5
2.3 **NH-Boost.DT** . . . 5
2.3.1 The algorithm . . . 5
2.3.2 The theoretical performance of **NH-Boost.DT** . . . 6
2.4 **Squint-Boost** . . . 6
2.4.1 **Squint** and second order quantile bounds . . . 6
2.4.2 **Squint-Boost** . . . 7

3 Experiments **8**
3.1 Setup . . . 8
3.2 Results . . . 8
3.2.1 **AdaBoost** . . . 8
3.2.2 **NH-Boost.DT** . . . 10
3.2.3 **Squint-Boost** . . . 12

4 Conclusion and future work **14**

Chapter 1

Introduction

1.1 Outline

1.1.1 Setting

There are three contexts with which we will be concerned in this thesis: classification, boosting and hedging. In the classification setting we are expected to produce the correct answer based on some input, while being provided with example problems and their correct answers so we can try to discover the hidden rules. The second setting is boosting which is an answer to the classification problem. Here we start out with a rule of thumb that must perform only slightly better than random guessing. We then proceed to improve the accuracy of this rule of thumb by trying to identify the important examples and adjusting our rule of thumb accordingly. The final context is that of hedging, meaning "to protect oneself against loss by making balancing or compensating transactions". Hedging is all about cleverly allocating resources to minimize losses. Hedging algorithms are often used as subroutines of boosting algorithms to find the difficult examples of a problem in an attempt to improve accuracy.

1.1.2 Scope

The main goal of this thesis is to compare the practical performance of two previously known algorithms: **AdaBoost**, **NH-Boost.DT** and one new algorithm called **Squint-Boost**. These are boosting algorithms based on hedge allocation algorithms called **Hedge**, **NormalHedge.DT** and **Squint** respectively. In [4] Koolen and Van Erven proved that **Squint** theoretically outperforms both **Hedge** and **NormalHedge.DT**. However, since the theory must account for edge cases which might not occur in the average scenario, it is possible that the difference in practical performance between **Squint** and the other two algorithms is negligible or, as will turn out to be the case, **Squint-Boost** might not outperform the previous algorithms at all. In this thesis we will implement all of the above algorithms to compare their accuracy as a function of the number of iterations over two data sets.

In this thesis we will only explore the algorithms in a practical setting but since **Squint** was very recently developed, the curious reader might wonder how **Squint-Boost** performs theoretically. In [6] Otten performs a similar exploration of these algorithms but from a theoretical perspective, exploring how **Squint-Boost** performs theoretically in comparison to the other two.

1.1.3 Basic structure

In chapter 2 we will give a formal description of the settings and procedures we will use throughout the thesis. We will also introduce the notations and basic terminology, as well as briefly discuss the results of previous works, both practical and theoretical. In chapter 3 we will discuss the results of the algorithms and compare their accuracy.

Chapter 2

The three boosting algorithms

2.1 Settings

We will now provide a formal description of the settings used in this thesis.

2.1.1 Classification

The first setting is that of *classification* [3]. In this setting we are presented with training examples and their correct label. We must then study these examples and come up with a rule to classify future unlabelled inputs. Here we have a finite number of labels lacking any structure such as order or magnitude. We will primarily concern ourselves with binary classification, i.e. results being either negative or positive, since the generalization to multi-class problems has been studied in [2] and is outside the scope of this thesis. An example of this setting is spam detection. Given enough correctly labelled example emails, we must formulate a rule to label new incoming emails as either genuine or spam.

In this setting we have an input and a label space, denoted $\mathcal{X}$ and $\mathcal{Y}$ respectively. For the remainder of this thesis we will generally set $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \{0, 1\}$ or $\mathcal{Y} = \{-1, 1\}$, the latter choice depending on convenience. Furthermore vectors will be denoted in boldface and their components will be denoted by a subscript k. To go back to our spam example, the labels might be 1 if the email is genuine and -1 if it is spam, while a feature vector $\mathbf{X} \in \mathcal{X}$ might consist of the number of exclamation marks and the number of dollar signs as its features. We will denote our data as follows:

$$\begin{pmatrix} Y_1 \\ \mathbf{X}_1 \end{pmatrix}, \ldots, \begin{pmatrix} Y_i \\ \mathbf{X}_i \end{pmatrix}, \ldots, \begin{pmatrix} Y_N \\ \mathbf{X}_N \end{pmatrix}$$

where i is the ith example ranging from 1 to N. Thus, to summarize: $X_{i,k}$ is the kth component of the ith example $\mathbf{X}_i$.

2.1.2 Hedging

In the context of hedging we will have to allocate our resources according to several strategies, in such a way that we minimize our losses. Solutions to hedge settings are often used as subroutines by boosting algorithms to identify important examples and to focus on those. An example of this process would be the following: imagine you are betting on a horse race. You have several friends, all of whom advise you according to some rule of thumb and you want to bet a fixed amount on every round. If you knew ahead of time which rule of thumb would be the best, you would bet everything according to that rule but this is usually not known. You try to distribute your betting money in such a way that your losses will not be much worse than if you bet everything according to your luckiest friend's advice.

Formally this means that an agent A has access to N *strategies*, often also called *experts*. It will then attempt to choose a distribution $\mathbf{p}^t$ over these strategies at every round, for T rounds. Since all

algorithms are iterative, we will use a superscript t to denote instances at time (or iteration) t. After choosing $\mathbf{p}^t$ the agent will receive a loss vector which we will denote as $\boldsymbol{\ell}^t$, and suffer loss equal to

$$\sum_{i=1}^{N} p_i^t \ell_i^t = \mathbf{p}^t \cdot \boldsymbol{\ell}^t.$$

In this context we will denote the total loss of the ith strategy as

$$L_i^T := \sum_{t=1}^{T} \ell_i^t$$

and the total loss of A as

$$L_A^T := \sum_{t=1}^{T} \mathbf{p}^t \cdot \boldsymbol{\ell}^t.$$

Our goal is to minimize what is called the *regret* which is defined as

$$R_A^T := L_A^T - \min_i L_i^T$$

i.e. the difference in the suffered loss and the minimal loss we could have had. Finally we will sometimes refer to the regret with respect to an expert i which will be denoted as

$$R_i^T := \sum_{t=1}^{T} r_i^t := \sum_{t=1}^{T} \mathbf{p}^t \cdot \boldsymbol{\ell}^t - \ell_i^t$$

2.1.3 Weak learners

In the context of boosting we will require algorithms which are called *weak learners* [2]. We will furthermore use **WeakLearn** to denote a generic weak learning algorithm. These weak learners are simply learning algorithms that produce hypotheses which must have an error rate on the training set of at most $\frac{1}{2} - \gamma$ for some $\gamma > 0$. This training error will then be expected to generalize to a hopefully low generalization error. The generalization error refers to the error on a set of examples which have their correct labels attached but were not used to train the algorithm but to test it after the training was done. This is to avoid something called *overfitting* and refers to the phenomenon that the rules constructed by the algorithms are overly complicated, which leads to the algorithm's performing very well on the training data but not on new data.

In this thesis the role of the weak learners will mostly be fulfilled by what are called *decision stumps*. These are decision trees of depth one i.e. the learner will measure a single feature of an input and then attempt to formulate an answer. Formally, this means that, given an input $\mathbf{X}$ the stump will attempt to find an optimal r and k such that the hypothesis $h : \mathcal{X} \to \mathcal{Y}$ given by

$$h(\mathbf{X}) := \begin{cases} 1 & \text{if } X_k \leq r \\ -1 & \text{otherwise} \end{cases}$$

has minimal error on our training set. Here we have chosen to check whether $X_k \leq r$ instead of $X_k \geq r$ but this is another choice for which the stump can optimize.

2.1.4 Boosting

Recall that the goal of boosting is to take a simple weak learner that only performs slightly better than random guessing and repeatedly assign weights to the examples so that after training, several instances of the weak learner will form a committee which has a significantly better performance than the original learner. This leads to the following formal setting: we require a set of labelled training data and a weak learning algorithm **WeakLearn**. Often a boosting algorithm can also accept a distribution on the training data to exploit any prior knowledge one might have, which is set to the uniform distribution if no extra knowledge is provided. The procedure at each iteration is to fit **WeakLearn** to the training

data which provides us with an hypothesis $h^t : \mathcal{X} \to \mathcal{Y}$. Using this hypothesis we will calculate a loss vector $\boldsymbol{\ell}^t$. This loss vector is a measure for how well **WeakLearn** performed on the training data and is usually some form of the training error. For example **AdaBoost** uses the weighted error $\mathbf{p}^t \cdot \boldsymbol{\ell}^t$ whereas **NH-Boost.DT** also uses a measure called the margin and its distribution, which we will define in section 2.3.1. Using this loss vector we will slightly alter the training set by attaching weights, which measure the relative importance of the examples, and then repeat the process. Our final hypothesis will then be some weighted majority vote among all of the hypotheses.

2.2 AdaBoost

Algorithm 1 AdaBoost

Require:

N labelled samples $\begin{pmatrix} y_1 \\ \mathbf{x}_1 \end{pmatrix}, \ldots, \begin{pmatrix} y_N \\ \mathbf{x}_N \end{pmatrix}$ with $\mathcal{Y} = \{0, 1\}$

Distribution D over the N examples

Weak learning algorithm **WeakLearn**

Number of trials T

1: **procedure AdaBoost**

2: **Initialize** the weight vector $w_i^1 = D(i)$ for $i = 1, \ldots, N$

3: **for** $t = 1, 2, \ldots, T$ **do**

4: Set

$$\mathbf{p}^t = \frac{\mathbf{w}^t}{\sum_{i=1}^N w_i^t}$$

5: Call **WeakLearn** ($\mathbf{p}^t$) and receive hypothesis $h^t : \mathcal{X} \to \mathcal{Y}$

6: Calculate the error of $h^t : \varepsilon^t = \sum_{i=1}^N p_i^t |h^t(\mathbf{x}_i) - y_i|$

7: Set $\beta^t = \varepsilon^t / (1 - \varepsilon^t)$

8: Update the weights

$$w_i^{t+1} = w_i^t (\beta^t)^{1 - |h^t(\mathbf{x}_i) - y_i|}$$

9: **end for**

10: **return**

$$h_f(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_{t=1}^T (\log 1/\beta^t) h^t(\mathbf{x}) \geq \frac{1}{2} \sum_{t=1}^T \log 1/\beta^t \\ 0 & \text{otherwise} \end{cases}$$

11: **end procedure**

2.2.1 The algorithm

We will now give a formal discussion of the **AdaBoost** [2] algorithm, which can be found in algorithm 1 above. The goal of the algorithm is to minimize the weighted error with respect to the calculated distribution over the training data. In the first step the algorithm will obtain a new distribution by normalizing the weights from the previous step, or from the initial distribution. This distribution will then be provided to **WeakLearn** which will form an hypothesis accordingly. When the hypothesis is formulated, the algorithm will calculate its error and set the parameter β, which can be interpreted as the learning speed. In the final step the algorithm will update the weights appropriately. It is important to observe that **WeakLearn** produces an hypothesis at every iteration and that the final hypothesis is essentially a weighted majority vote among all of these hypotheses.

Observing the **Hedge** [2] algorithm alongside **AdaBoost**, one can see, most notably in the way the weights are updated, that **Hedge** is used as a subroutine. Essentially **AdaBoost** treats the weighted error of the current hypothesis as the regret and uses **Hedge** as a way to identify the difficult examples and focus **WeakLearn** on those examples. We will see constructions similar to this one appear with the other algorithms, which we will discuss in their respective sections.

2.2.2 The theoretical performance of AdaBoost

Here we will discuss the theoretical performance of **AdaBoost**. One of the main results from [2], is the following theorem:

Theorem 2.2.1. *[2] Suppose **AdaBoost** runs for T rounds calling the weak learning algorithm **WeakLearn** which generates hypotheses with errors $\varepsilon^1 \dots, \varepsilon^T$ (as defined in step 6 in algorithm 1), then the training error $\varepsilon := \sum_{i=1}^{N} D(i)[h_f(x_i) \neq y_i]$ of the final hypothesis h_f output by **AdaBoost** is bounded above by*

$$\varepsilon \leq 2^T \prod_{t=1}^{T} \sqrt{\varepsilon^t(1-\varepsilon^t)}$$

We will omit the proof, since it is not the focus of this thesis but we will briefly discuss its consequences. As discussed above one might find it strange that every hypothesis is allowed a vote in the final hypothesis. This is still, however, an improvement over previous algorithms since the final error will now depend on the error of all hypotheses instead of only the worst, as was the case in previous works. It is also not immediately obvious that this upper bound does us any good since it contains exponents and products for which it is not immediately apparent how they cancel out. In their paper [2], Freund and Shapire show the above stated upper bound implies the following:

$$\varepsilon \leq \exp\left(-2\sum_{t=1}^{T}(\gamma^t)^2\right)$$

where the γ^t is defined such that $\varepsilon^t = 1/2 - \gamma^t$ holds. This upper bound is more useful since it decreases exponentially, leading us to conclude that **AdaBoost** has a good theoretical performance.

2.3 NH-Boost.DT

2.3.1 The algorithm

In their paper [5] Luo and Schapire use the framework they had previously developed for drifting game analysis to develop a new algorithm called **NormalHedge.DT** which improves upon the previous **NormalHedge** algorithm. Whereas **NormalHedge** depends on a numerical search to find the right parameters, Luo and Schapire first develop a general setting for Hedge algorithms which requires convex functions as input and they then proceed to derive a parameter free version which they called **NormalHedge.DT**. This algorithm was much more computationally efficient and was easier to generalize to other settings such as boosting. They then proceeded to generalize this to several settings including boosting, which led them to derive the **NH-Boost.DT** algorithm which we will discuss here.

Algorithm 2 NH-Boost.DT

Require:

N labelled samples $\begin{pmatrix} y_1 \\ \mathbf{x}_1 \end{pmatrix}, \dots, \begin{pmatrix} y_N \\ \mathbf{x}_N \end{pmatrix}$ with $\mathcal{Y} = \{-1, +1\}$

Weak learning algorithm **WeakLearn**

Number of trials T

1: **procedure NormalHedge.DT**
2: **Initialize** $\mathbf{s}_0 = \mathbf{0}$
3: **for** $t = 1, 2, \dots, T$ **do**
4: Set: $\mathbf{p}^t \propto \exp([\mathbf{s}^{t-1} - 1]_-^2/3t) - \exp([\mathbf{s}^{t-1} + 1]_-^2/3t)$
5: Call **WeakLearn**$(\mathbf{p}^t)$ and get hypothesis h^t with edge $\gamma^t = \frac{1}{2}\sum_{i=1}^{N} p_i^t y_i h^t(\mathbf{x}_i)$
6: Set: $\mathbf{s}^t = \mathbf{s}^{t-1} + \frac{1}{2} y_i h^t(\mathbf{x}_i) - \gamma^t$
7: **end for**
8: **return** $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} h^t(\mathbf{x})\right)$
9: **end procedure**

2.3.2 The theoretical performance of NH-Boost.DT

Again **NH-Boost.DT** uses several instances of **WeakLearn**, which are all expected to have *edge* at least γ, i.e. have error $\frac{1}{2} - \gamma$ with respect to the training distribution, to form a committee which decides a final output based on a majority vote. In their paper they formulated and proved the following theorem:

Theorem 2.3.1. *[5] After T rounds the training error of **NH-Boost.DT** is of order $\mathcal{O}(\exp(-\frac{1}{3}T\gamma^2))$ and the fraction of training example with margin at most $\theta(\leq 2\gamma)$ is of order $\mathcal{O}(\exp(-\frac{1}{3}T(\theta - 2\gamma)^2))$*

We will again omit the proof but briefly discuss its consequences here. Firstly it is useful to observe that the training error of **NH-Boost.DT** is comparable to that of **AdaBoost**. However, instead of looking only at the training error and trying to generalize that to the generalization error, Luo and Schapiere also make statements with respect to the margin distribution which is a measure of confidence often used in classification settings. The margin of an example $(\mathbf{x}, y)$ with respect to some weak learner is defined as $\frac{1}{T}\sum_{t=1}^{T} yh^t(x)$ i.e. the difference between the fractions of correct and incorrect hypotheses of the example. Here its magnitude measures the confidence of the committee and its sign measures whether or not the weak learner was correct. In this light the above theorem also states that the fraction of training examples the algorithm deems important enough also decreases exponentially and thus it ignores many of the examples every round, making it much more computationally efficient.

As we discussed before **AdaBoost** uses **Hedge** as a subroutine. Similarly **NH-Boost.DT** uses **NormalHedge.DT** as a subroutine by feeding it a loss vector which is based on the margins of the hypothesis of the previous rounds. So, in essence the main difference between **NH-Boost.DT** and **AdaBoost** is that they use different hedging algorithms as subroutines but the overall (boosting) framework remains the same.

2.4 Squint-Boost

Unlike the previous sections, here we will first take a more careful look at **Squint**, which is the hedging algorithm instead of the boosting algorithm. This is because **Squint** has not been studied in the context of boosting before and therefore we wish to introduce it formally before we discuss it in the context of boosting.

2.4.1 Squint and second order quantile bounds

In their paper [2] Freund and Shapire proved that **Hedge** can achieve

$$R_i^T < \sqrt{T \ln N} \qquad \text{for each expert } i.$$

While this is a reasonable worst-case performance, one can ask if we can do better in the common case. Furthermore, aside from the performance, the algorithm has another major drawback, namely that the bound increases if one adds experts, even ones that perform well. So, even when one adds experts that perform well and one cannot eliminate other experts that do not, the theoretical performance of **Hedge** still decreases. In answer to this problem two lines of research were started. The first one attempted to develop algorithms which perform well with respect to a so called *quantile bound*, i.e. a bound on the regret that considers a subset of experts instead of just one. The second line of research attempted to improve the bound's dependence on T to a dependence of some form of cumulative variance (or some related second-order quantity) which is often significantly smaller than T.

In response to this research Koolen and Van Erven introduced **Squint** [4] (for second order quantile integral). They bring several versions of the algorithm forward which use different priors on the experts, but the best results came from the use of an improper prior. Using this prior **Squint** achieves both a quantile bound and a variance guarantee and we will use this version throughout the thesis. Using the notations $R_{\mathcal{N}}^T := \mathbb{E}_{\pi(i|\mathcal{N})} R_i^T$ and $V_{\mathcal{N}}^T := \mathbb{E}_{\pi(i|\mathcal{N})} V_i^T$ to denote the average regret under the prior among the reference experts and the uncentred variance of the excess losses respectively with $V_i^T := \sum_{t=1}^{T} (r_i^t)^2$ and $\mathcal{N}$ denoting any subset of experts, they formulate the following theorem:

Theorem 2.4.1. *[4] Let $\mathcal{N}$ be any subset of experts and let **Squint** run for T rounds using the update rule*

$$w_i^{t+1} \propto \pi(i) \int_0^{\frac{1}{2}} e^{\eta R_i^t - \eta^2 V_i^t} \, d\eta = \pi(i) \frac{\sqrt{\pi} e^{\frac{(R_i^t)^2}{4V_i^t}} \left(\operatorname{erf}\left(\frac{R_i^t}{2\sqrt{V_i^t}} \right) - \operatorname{erf}\left(\frac{R_i^t - V_i^t}{2\sqrt{V_i^t}} \right) \right)}{2\sqrt{V_i^t}}$$

*then the regret of **Squint** is bounded above by*

$$R_{\mathcal{N}}^T \leq \sqrt{2V_{\mathcal{N}}^T} \left(1 + \sqrt{2 \ln\left(\frac{\frac{1}{2} + \ln(T+1)}{\pi(\mathcal{N})} \right)} \right) + 5 \ln\left(1 + \frac{1 + 2\ln(T+1)}{\pi(\mathcal{N})} \right)$$

Again we will omit the proof but briefly discuss it. Firstly it should be observed that the fact that we have a closed-form for the integral which makes it computationally efficient. Furthermore it is usefully to note that we consider $\ln(\ln(x))$ to be essentially constant so the above bound indeed achieves a quantile bound with a dependence on the variances, improving the bounds of previous algorithms.

2.4.2 Squint-Boost

Plugging **Squint** into the boosting framework yields **Squint-Boost** which can be found in algorithm 3 below. Since not much is yet known about **Squint-Boost** we will not go into great detail here, as we will examine it and its practical perfomance in chapter 3 and the theoretical performance is outside the scope of this thesis. A more in-depth theorectical comparison and exploration of the algorithms was performed by Otten in [6].

Algorithm 3 Squint-Boost

Require:

N labelled samples $\binom{y_1}{\mathbf{x}_1}, \ldots, \binom{y_N}{\mathbf{x}_N}$ with $\mathcal{Y} = \{-1, +1\}$

Weak learning algorithm **WeakLearn**

Number of trials T

1: **procedure Squint-Boost**
2: **Initialize** $\mathbf{R^0}, \mathbf{V^0}, \mathbf{r^0} = \mathbf{0}$
3: **for** $t = 1, 2, \ldots, T$ **do**
4: Set: $p_i^t \propto \pi(i) \frac{\sqrt{\pi} e^{\frac{(R_i^t)^2}{4V_i^t}} \left(\operatorname{erf}\left(\frac{R_i^t}{2\sqrt{V_i^t}} \right) - \operatorname{erf}\left(\frac{R_i^t - V_i^t}{2\sqrt{V_i^t}} \right) \right)}{2\sqrt{V_i^t}}$
5: Call **WeakLearn**($\mathbf{p}^t$) and get hypothesis h^t with edge $\gamma^t = \sum_{i=1}^N p_i^t h^t(\mathbf{x}_i) y_i$
6: Set: $r_i^t = \frac{1}{2} h^t(\mathbf{x}_i) y_i - \gamma^t$
7: Set: $R_i^t = R_i^{t-1} + r_i^t$
8: Set: $V_i^t = V_i^{t-1} + (r_i^t)^2$
9: **end for**
10: **return** $H(\mathbf{x}) = \operatorname{sign}\left(\sum_{t=1}^T h^t(\mathbf{x}) \right)$
11: **end procedure**

Chapter 3

Experiments

3.1 Setup

In this chapter we will examine the practical relative performance of the three algorithms. We will do so in several ways. The first way is with the use of simulated data (recreated from [3]). We have ten features $X_1, \dots, X_{10}$ which are drawn from a standard independent Gaussian. Their label is determined (deterministically) by the following rule:

$$Y := \begin{cases} 1 & \text{if } \sum_{k=1}^{10} (X_k)^2 > \chi^2_{10}(0.5) \\ -1 & \text{otherwise} \end{cases}$$

Where $\chi^2_{10}(0.5) = 9.34$ is the median of a chi-squared random variable with ten degrees of freedom. This is to ensure that there are roughly the same number of labels in each category.

The second way we will test the algorithms is the UCI “a9a Adult data set” [1]. Here the goal is to predict whether an individual will earn more than $50,000 per year or not. The original data set has 14 features, among which six are continuous and eight are categorical. In this data set, continuous features are discretized into quantiles, and each quantile is represented by a binary feature. Furthermore a categorical feature with m categories is converted to m binary features.

In an attempt to keep the comparison as consistent as possible we will, in all settings, use 32,561 observations for training and 16,281 for testing. This is because these are the number of observations in the a9a data set. This ratio of training to testing observations is not uncommon. Recall to avoid overfitting we cannot test the algorithm on the same data set we use for training. Thus any observations which are used to test cannot be used to train. Therefore this kind of ratio is desirable.

We implemented the algorithms in Python 3. Here we used Scikit-learn’s implementation of a decision tree as our weak learner [7]. For the interested reader our implementations of all the algorithms discussed in this thesis, as well as everything necessary to recreate the experiments, is freely available at `https://github.com/dvente/bscthesis`

3.2 Results

We will now discuss the results of our experiments in order to compare the algorithms.

3.2.1 AdaBoost

Figure 3.1 shows the generalization error of **AdaBoost** as a function of the number of iterations on the simulated data set. Here we see it works as one would expect. The decision stump initially performs with an error rate of 47% which is indeed only slightly better than random guessing. **AdaBoost** outperforms this and even a large tree with 325 nodes and an error rate of 30% after just tens of iterations and reaches an error rate as low as 7.7% after 500 iterations.

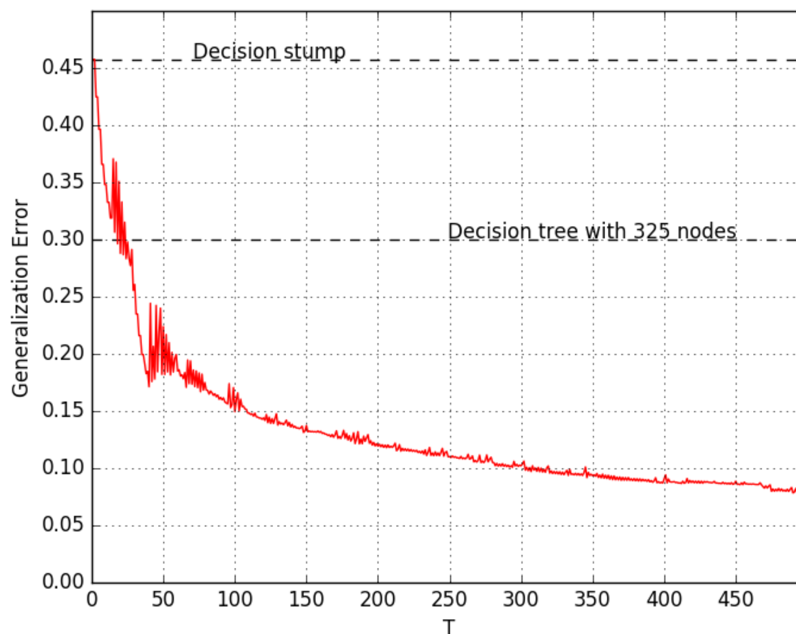

Figure 3.1: Generalization error of AdaBoost as a function of the number of iterations over the simulated data set. The error rate of the decision stump and a decision tree with 325 nodes are shown for reference

Figure 3.2 again shows the generalization error of **AdaBoost** as a function of iterations but this time over the a9a data set. Here the situation is somewhat different. One sees that the algorithm still works but here we run into some limitations. As we can see, the initial stump performs significantly better to start with, with an error rate of 23.6%. After just 50 iterations we see diminishing returns set in quite drastically, seeing only a slight improvement from 15.7% to 15.2% over the course of 450 iterations. This also happens with the simulated data but to a much lesser degree. This is probably due to the fact that the simulated data is much more homogeneous, where in fact all features are equally important, with the difficult examples being those where all features are small but barely meet the criteria. The a9a data however most likely has a few features that considerably improve the accuracy (for example education), but beyond that, the data does not show any important examples so that the gains, after the low-hanging fruits have been picked, are marginal. While this is still a good result it is useful to keep this phenomenon in mind while we examine the remaining algorithms. This characteristic highlights a fundamental disadvantage of a weak learner, namely that depending on the data set there might simply not be a way to improve the accuracy beyond a certain point.

A final observation on the reference trees should be useful here: while we tried to keep the size of the trees as consistent as possible, it is quite difficult to enforce a specific size over different data sets, especially ones that differ as strongly as our two data sets do. This is why the trees have slightly different sizes across the different plots. We attempted to keep the sizes as similar as possible by enforcing a maximum depth of $\lfloor \log_2(500) \rfloor = 8$, letting the algorithms fit the trees as best possible under this constraint. We have chosen this because it ensures an upper bound of 500 nodes, which we found to be a reasonable bound for the size and the shapes of our considered data.

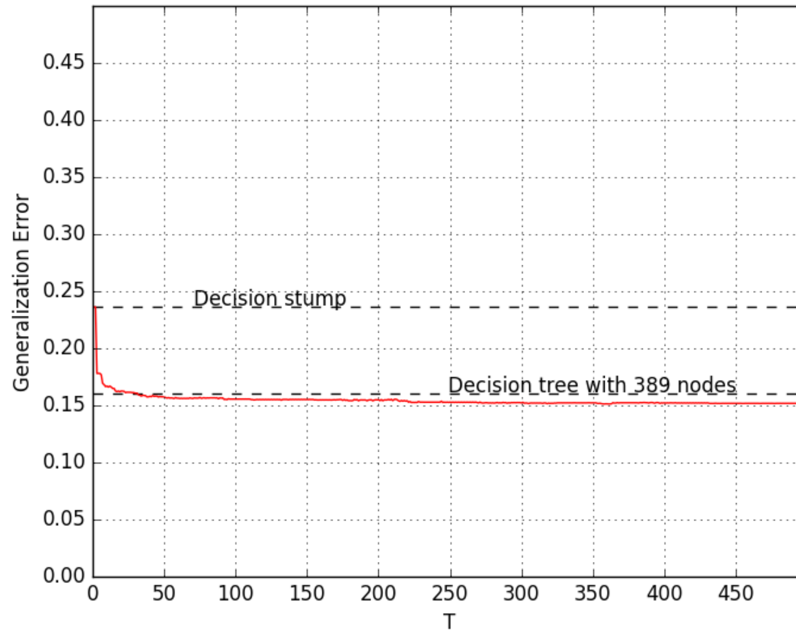

Figure 3.2: Generalization error of AdaBoost as a function of the number of iterations over the a9a data set. The error rate of the decision stump and a relatively large decision tree are again shown for reference

3.2.2 NH-Boost.DT

Before we discuss the results of **NH-Boost.DT** here we would like to highlight two important differences between **NH-Boost.DT** and **AdaBoost**. The first is that to improve performance **NH-Boost.DT** attempts to assign zero weight to unimportant examples in order to improve performance. The percentage of examples which were assigned weight zero is shown in green in the plots below. The second important difference is that whereas **AdaBoost** decides the label based on a reference threshold, **NH-Boost.DT** simply takes an unweighted majority vote. A consequence of this is that on the even iterations the final committee might be tied on a decision. In an attempt to reflect this in our comparison we count the inconclusive tests (shown in a percentage of the testing size in blue) and recorded these as “half a mistake” since the committee was neither right nor wrong. Of course this impacts the performance of the algorithm quite significantly as can be seen below. One important remark is that the committees are never tied in the case of an odd size. For the sake of clarity the graphs of the inconclusive tests in the plots below only show the percentage on the even iterations, those on the odd iterations always being 0.

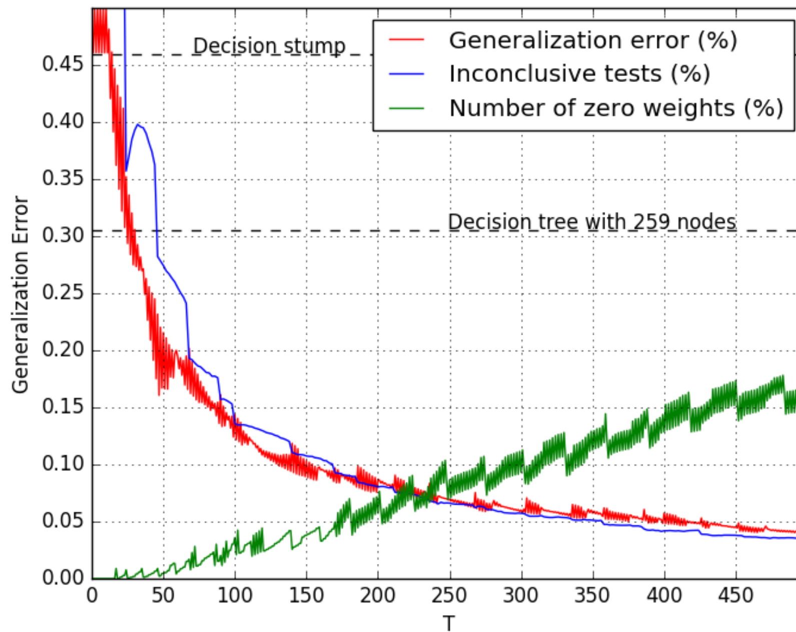

Figure 3.3: Generalization error of **NH-Boost.DT** as a function of the number of iterations on the simulated data. The error rates of trees with 3 and 259 nodes respectively are again shown for reference. It is important to remark that for the sake of clarity we omitted the inconclusive data points of all odd iterations as they are always 0.

Here again figure 3.3 depicts the generalization error of **NH-Boost.DT** as a function of the number of iterations on the simulated data set. The first thing one notices is that this mimics the behaviour predicted by the theory, as well as the behaviour of **AdaBoost**. The error rate as well as the percentage of inconclusive tests steadily declines whilst the percentage of zero weights steadily increases. In terms of actual percentages the algorithm performs quite well compared to **AdaBoost**. Where **AdaBoost** achieved an error rate of 7.7% after 500 iterations **NH-Boost.DT** achieves 3.9% which is much better. This is achieved with a zero weight percentage of 15.7% at the end. This, it should be noted, is a very significant portion.

When looking at the a9a data the situation is also a bit different. Figure 3.4 again shows the generalization error of **NH-Boost.DT** as a function of the number of iterations but over the a9a data set. The first obvious point to note when looking at figure 3.4 is that the algorithm is (initially) quite confused about the number of examples to assign a zero weight. One possible cause is that the weights of the previous round are determined by the performance of the previous iteration and that the algorithm gets slightly confused by the inconclusive tests. The peaks of zero weights are almost all around the 28% whereas the last iteration has a zero weight percentage of 23.2%. This compares quite favourably to the previous data set, which can be explained by our interpretation that after the low hanging fruit has been picked many examples become irrelevant and thus get assigned zero weight. One can, however, also wonder how reliable this interpretation is due to all the peaks and valleys.

Furthermore we again see the “low-hanging fruit” characteristics we saw in the **AdaBoost** implementation. Here the final error rate bottoms out at 15.1% compared to **AdaBoost** 15.2% which is almost identical. One can wonder about the role the inconclusive tests play in this conclusion. This is not however of great concern, since the accuracy on the odd iterations is also almost identical. Furthermore, the inconclusive tests are an indication that the committee is not certain on those predictions even when they are not tied in the odd iterations. This tells us that these percentages are indeed very comparable.

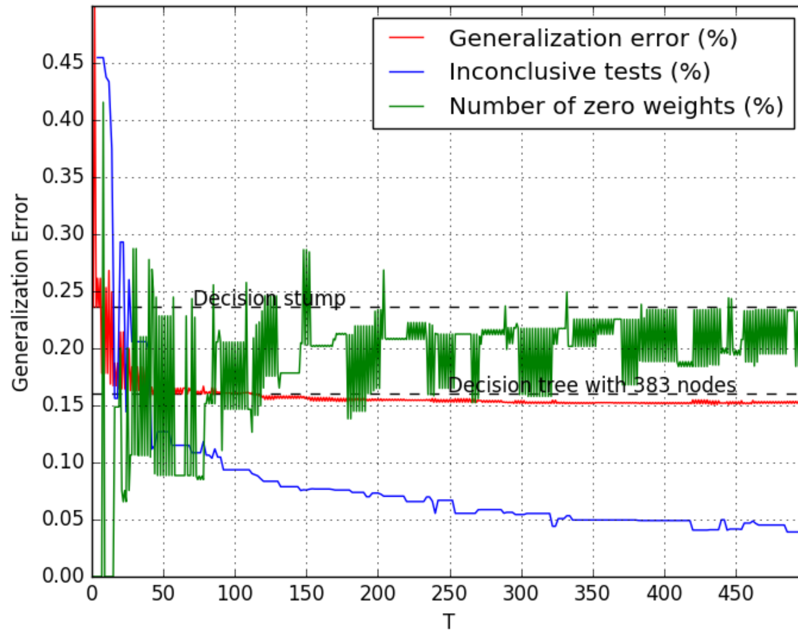

Figure 3.4: Generalization error of **NH-Boost.DT** as a function of the number of iterations on the a9a data. The error rates of trees with 3 and 383 nodes respectively are again shown for reference. It is important to remark that for the sake of clarity we omitted the inconclusive data points of all odd iterations as they are always 0.

3.2.3 Squint-Boost

Finally we will consider **Squint-Boost**. Figure 3.5 below shows the generalization error of **Squint-Boost** as a function of the number of iteration over the simulated data set. Looking at it we again see the predicted behaviour. After a few initial spikes, both the error rate and the fraction of inconclusive tests steadily decline. As one can see it indeed performs somewhat similar to the other algorithms, finally achieving an error rate of 9.2% compared to the 7.7% and 3.9% of **AdaBoost** and **NH-Boost.DT** respectively. While it is still a good result, we conclude that **Squint-Boost** performs worse than **NH-Boost.DT** and even **AdaBoost**.

Finally figure 3.6 depicts the generalization of **Squint-Boost** as a function of the number of iterations over the a9a data set. When looking at it, we again see the same behaviour as with the other two algorithms. The initial improvement is quite fast but it bottoms out fairly quickly, finally achieving an error rate of 15.1% which is again extremely similar to the 15.1% and 15.2% of the other two algorithms.

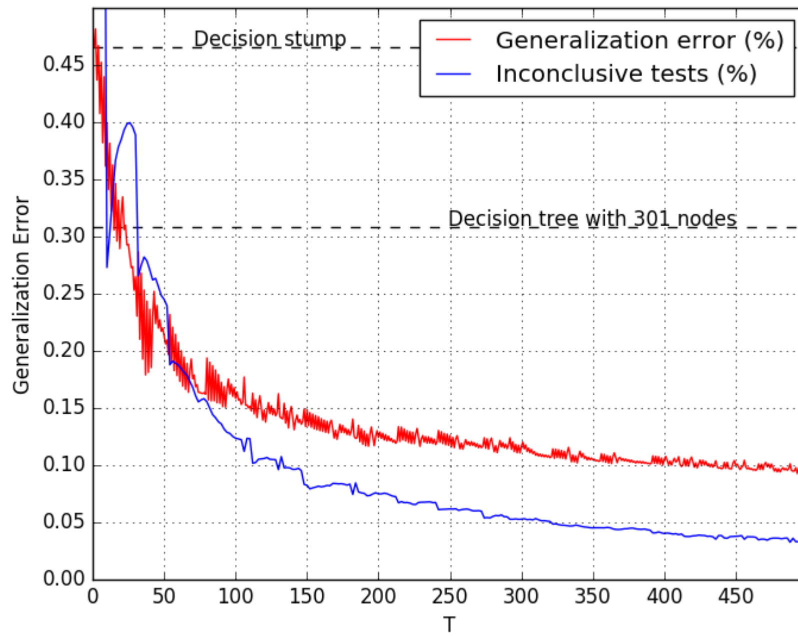

Figure 3.5: Generalization error of **Squint-Boost** as a function of the number of iterations on the simulated data. The error rates of trees with 3 and 301 nodes resp. are again shown for reference. It is important to remark that for clarities' sake we omitted the inconclusive data points of all odd iterations as they are always 0.

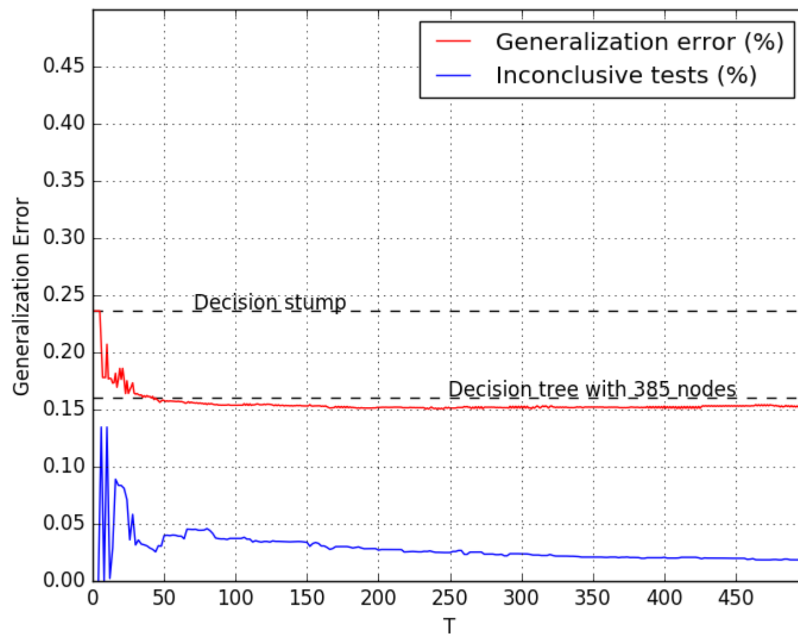

Figure 3.6: Generalization error of **Squint-Boost** as a function of the number of iterations on the a9a data. The error rates of trees with 3 and 385 nodes resp. are again shown for reference. It is important to remark that for clarities' sake we omitted the inconclusive data points of all odd iterations as they are always 0.

Chapter 4

Conclusion and future work

In this thesis we compared the practical performance of the three algorithms **AdaBoost**, **NH-Boost.DT** and **Squint-Boost**. We used both a simulated data set and a real data set to compare the algorithms. In the case of the simulated data set we saw a clear hierarchy, namely that **NH-Boost.DT** outperformed **AdaBoost** which in turn outperformed **Squint-Boost**. In the setting of the real data set there were no significant performance differences. We also illustrated a possible problem with boosting, namely that it might be possible that the data has a few obvious criteria which work fairly well but is very difficult to improve beyond those initial criteria, a property we called the low-hanging fruit property. In this case boosting might not be able to improve beyond a certain point whilst still increasing computation time in both the training and the predicting phase.

Because of time constraints we were only able to compare the algorithms on two data sets. Future work could include the testing over a broader range of data, with both small and large shapes, that may or may not have the low-hanging fruit property. In our hypotheses about the performance of **Squint-Boost** we assumed that the variance it uses would be smaller than the number of iterations; this need not be the case however. This could be an explanation why performed worse than the other two, and future work could include investigations on how this variance behaves and how this actually impacts performance. Furthermore it could try to devise certain tests to determine whether the data to be used is conducive to boosting. One of the problems **Squint-Boost** and **NH-Boost.DT** encounter is the problem of tied committees, so further work could include a search to break the ties in a meaningful way to potentially improve the accuracy of the algorithms and to see if this indeed works.

Bibliography

[1] LIBSVM Data: Classification (Binary Class). Availabel at https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/binary.html.

[2] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. System Sci.*, 55(1, part 2):119–139, 1997.

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning*. Springer Series in Statistics. Springer, New York, second edition, 2009. ISBN 978-0-387-84857-0. Data mining, inference, and prediction.

[4] Wouter M. Koolen and Tim van Erven. Second-order quantile methods for experts and combinatorial games. *Proceedings of The 28th Conference on Learning Theory*, 2015.

[5] Haipeng Luo and Robert E. Schapire. A drifting-games analysis for online learning and applications to boosting. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1368–1376. Curran Associates, Inc., 2014.

[6] Heleen Otten. A theoretical analysis of boosting algorithms, 2016. To appear at https://www.math.leidenuniv.nl/nl/theses/year/2016/.

[7] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.