RWTH Aachen University
Faculty of Mathematics, Computer Science and Natural Sciences
Chair of Computer Science 13 (Computer Vision)
Prof. Dr. Bastian Leibe

Seminar Report

A Brief Contextualisation of the Fundamentals of Boosting Algorithms

Freund and Schapire's AdaBoost Algorithm

Marawan Emara
Matriculation Number: 391003

February 2021

Advisor: Dan Jia

Contents

List of Figures

List of Tables

Abstract

In the field of machine learning and artificial intelligence, the training of algorithms plays an essential role. This sort of training could be achieved through an array of different manners, such as Logistic Regression, Decision Trees or Boosting. In this paper, I will focus on the field of boosting algorithms, specifically the AdaBoost, and their major role within machine learning. Moving along from introducing how boosting and the first boosting algorithm came to be, then to how the AdaBoost algorithm would be applied in real terms, all the way to more modern research work based on the original AdaBoost algorithm, which was introduced by Freund and Schapire in 1993. Throughout this paper, it will become quite evident how momentous the AdaBoost was, as well as how it still plays a considerable role in training algorithms and boosting classifiers.

1 Introduction

In their original papers, "A Recursive Majority Gate Formulation" [Sch90] and "Boost by Majority" [Fre95], Robert Schapire and Yoav Freund set out to answer a question posed by Michael Kearns in his 1988 publication, "Thoughts on Hypothesis Boosting". The question presented asked if "an efficient algorithm that outputs an hypothesis of arbitrary accuracy" [Kea88] exists. Separately, each of Freund and Schapire were able to develop such an algorithm, with Freund's "boost by majority" being the more efficient algorithm [FS97]. Nevertheless, both algorithms had problems set within them; for example, Freund's algorithm required knowledge of the bias of the weak learning algorithm beforehand. This presented a practical hurdle, since such a bias is seldom known to the user. Together, Freund and Schapire developed the AdaBoost as a way to tackle this problem, with the new algorithm being "very nearly as efficient as boost-by-majority" [FS97].

The new boosting algorithm, laid out in their paper "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting", is described as adaptive due to being able to adapt with the accuracy of each output given by the weak learning algorithm. This quality of the AdaBoost essentially resolves the the prerequisite of knowledge of the bias. Subsequently, the AdaBoost algorithm spawned new research into the field of boosting, wherein the algorithm continues to define that precise field and any further research done within it.

Throughout this paper, I will attempt to explain simply what the "on-line allocation algorithm" is, how the AdaBoost algorithm was later derived from it, and finally the AdaBoost's implementation as well as some related research. Initially, one must understand the fundamentals of boosting, as well as the terminology used within the field, as to be able to better understand the algorithm itself and its efficacy. Analogous with understanding the algorithm would be a simple walk-through of its applications, which would greatly improve one's comprehension of how the AdaBoost is intended to be used beyond the theoretical approach. Beyond all of that, and just to show how momentous the AdaBoost was to its field, I will showcase some related work that relied on Freund and Schapire's papers for their development.

2 Development of the First Boosting Algorithm

2.1 Fundamentals of Boosting

One of the more important contributions made in the field of machine learning was the paper written by Leslie Valiant in 1984, which lays out a general computational model for the analysis of a machine learning algorithm [Val84]. Valiant's "A Theory of the Learnable" sets a few fundamentals to assess the ability of a machine to learn, with the three main points concerning the provability to learn whole classes and classify them, the non-triviality of classes and their appropriateness, and finally the prerequisite of a "feasible number of steps" for the process [Val84]. These same basic principles are outlined within Freund and Schapire's "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting", in which they outline a weak PAC-learning[1] algorithm and how it could be "boosted" in order for it to become a stronger PAC-learning algorithm.

Moving on from PAC-learning, one must also comprehend what classification and classifiers entails within the field of machine learning and thus also boosting. Classifiers, at their most basic functions, are algorithms that help classify data by using certain inputs and giving out certain outputs. Fundamentally there exists three types of classifiers: random classifiers, weak classifiers and strong classifiers. Random classifiers classify the input into the output using completely random metrics; in which way there exists no real strategy or algorithm to the way the data is classified, and thus the probability that the output is correctly classified is random. On the other hand, weak classifiers perform better in terms of their probability of correctly classifying the data. They, however, do not perform better than a "coin toss", and therefore the probability does not equate or cross the 50% threshold. Finally, strong classifiers are able to classify data

[1]The abbreviation "PAC" is derived from the name "probably approximately correct", which outlines the two main principles set within Valiant's "A Theory of the Learnable"; namely the high probability with which a process is able to gain low generalisation error [Val84].

more accurately than both random and weak classifiers. The probability of a strong classifier ensuring that the given input is produced into the correct output lies between 50% and 100%.

Both PAC-learning and classifiers form the essentials of the boosting field. This type of terminology is repeatedly used within Freund and Schapire's paper, as well as in most subsequent research done on its basis.

2.2 Hedge Algorithm

For us to be able to derive the AdaBoost algorithm, we must first understand the initial algorithm set out for its derivation, namely Hedge(β). Hedge(β) was designed to deal with the on-line allocation problem, wherein it is later modified to deal with the boosting of weak classifiers and converting them into ones "with arbitrarily high accuracy" [FS97].

With the following parameters:
$\beta \in [0,1]$
initial weight vector $w^1 \in [0,1]^N$ with $\sum_{i=1}^N w_i^1 = 1$
number of trials T
For $t = 1,2,\dots,T$ **do**

1. Choose the allocation

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

2. Receive loss vector $\ell^t \in [0,1]^N$ from environment.
3. Suffer loss $p^t \cdot \ell^t$.
4. Set the new weights as

$$w_i^{t+1} = w_i^t \beta^{\ell_i^t} \tag{2}$$

Figure 1: The on-line allocation algorithm [FS97].

In the on-line allocation algorithm, Hedge(β), the user chooses the allocation, p^t, which is equivalent to the weights at trial T, divided by the sum of the total amount of weights. This allocation then suffer a loss, ℓ^t, which cannot go below a loss equivalent to zero and cannot go beyond a loss equivalent to one. Each new weight is then set for the next trial, where each weight is equivalent to the previous weight (the weight at t) multiplied by the β[2] of the loss for each weight at t.

At heart, what the Hedge(β) does is that it initially gives all the classifiers the same initial weight. With each trial, or step, that the algorithm goes through, the weights of the least efficient classifiers would suffer heavier losses than the more efficient classifiers. This gives more efficient classifiers a bigger say in the final result, also called the final hypothesis. Hedge(β), also called the on-line allocation algorithm, places the foundation on which the adaptive boosting algorithm is developed, which we will discuss in Sec. 2.3.1, concerning the derivation of the AdaBoost algorithm.

[2] β is chosen by the user as to "maximally exploit any prior knowledge we may have about the specific problem at hand" [FS97], as discussed in Sec. 2.2 of Freund and Schapire's paper, where they also show how it is chosen.

2.3 AdaBoost Algorithm

2.3.1 Derivation of the AdaBoost Algorithm Using the Hedge Algorithm

In Sec. 2.3.1, I will briefly describe how the on-line allocation algorithm, which was described in Sec. 2.2, would then be modified to produce the adaptive boostung algorithm. It is also essential to describe what a **WeakLearn** algorithm is, in order for Fig. 2 to be as clear as possible.

Using the defining principles of PAC-learning, as well as the definition of classifiers, which was mentioned in Sec. 2.1, Freund and Schapire set the **WeakLearn** as a weak PAC-learning algorithm, thus containing a generic set of weak classifiers. This weak PAC-learning algorithm is transformed using **AdaBoost** into a stronger PAC-learning algorithm through adaptive boosting.

The algorithm for the AdaBoost is set as follows:

With the following input:
    sequence of N labelled examples $\langle (x_1, y_1), \ldots, (x_N, y_N) \rangle$
    distribution D over the N examples
    weak learning algorithm **WeakLearn**
    integer T specifying the number of iterations
Initialise the weight vector: $w_i^1 = D(i)$ for $i = 1, \ldots, N$.
For $t = 1, 2, \ldots, T$ **do**

1. Set

$$p^t = \frac{w^t}{\sum_{i=1}^{N} w_i^t} \tag{3}$$

2. Call **WeakLearn**, providing it with the distribution p^t; get back a hypothesis $h_t : X \longrightarrow [0, 1]$.
3. Calculate the error of $h_t : \epsilon_t = \sum_{i=1}^{N} p_i^t |h_t(x_i) - y_i|$.
4. Set $\beta_t = \epsilon_t / (1 - \epsilon_t)$.
5. Set the weights vector to be

$$w_i^{t+1} = w_i^t \beta_t^{1-|h_t(x_i) - y_i|} \tag{4}$$

6. **Output** the hypothesis

$$h_f(x) = \begin{cases} 1 & \text{if } \sum_{t=1}^{T} (log 1/\beta_t) h_t(x) \geq \frac{1}{2} \sum_{t=1}^{T} log 1/\beta_t \\ 0 & \text{otherwise.} \end{cases} \tag{5}$$

Figure 2: The modified on-line allocation algorithm, which creates the AdaBoost algorithm [FS97].

The algorithm shown in Fig. 2 is, at first glance, very similar to the one set in Fig. 1. That is due to the AdaBoost algorithm being derived from and built off of the on-line allocation algorithm. What the AdaBoost algorithm does starts off very similar to Hedge(β); wherein the classifiers' weights are all intialised to be equivalent using $\frac{1}{\sum_{i=1}^{N} w_i^1}$. Following that, the distribution p^t is set in the same way as the Hedge(β) algorithm, where it is then provided to the weak PAC-learning algorithm, **WeakLearn**, and the output, also called the hypothesis, taken. The error for the aforementioned output is then calculated. Notice this step is where the algorithm receives its "adaptive" quality, where the error is calculated for the weak PAC-learning algorithm in each iteration independently. Afterwards, in the fourth step of the process, β for the concurrent iteration is set. This diverges a bit from the Hedge(β) algorithm, where β is chosen by the user depending on acquired knowledge of the algorithm. The next step, step five, is also very similar to the on-line allocation algorithm, where in the classifiers' weights for the next iteration are set by multiplying the current weights with β. Here, however, the error calculated in step three is further taken into account. Finally, the final hypothesis, or output, is then made through a concatenation of the previous steps and the decisions made by their classifiers.

All of this helps the adaptive boosting algorithm develop the weak PAC-learning algorithm into an algorithm with "one with arbitrarily high accuracy" [FS97]. Nonetheless, the AdaBoost remains sensitive to outliers and noisy data sets. This problem would then be tackled down the line, with the development of new algorithms based on Freund's "boost by majority" or the AdaBoost, as can be seen in Sec. 4.3 concerning the BrownBoost. Furthermore, the AdaBoost algorithm itself has a few variations that were described in Freund and Schapire's paper, which essentially work on different types of data sets, such as multi class, single label data sets or multi class, multi label data sets. These variations are further discussed in the upcoming Sec. 2.3.2.

2.3.2 Variants of the AdaBoost Algorithm

There are two sets of variants made to the original AdaBoost algorithm which were also presented in the same paper as the AdaBoost itself; one variant is an extension of the AdaBoost that includes multi class problems and the other is an extentions that includes multi class, multi label problems. Multi class problems simply imply that a problem is of a higher level than just a binary problem, where the solution could be one of multiple (more than one) solutions. Meanwhile, multi label problems include non-mutually-exclusive classifications, meaning that they are related in some regard and not completely discernible from one another. These two major variations help include a larger variety of problems, in which the AdaBoost is able to carry out its task of boosting the algorithm.

In regards to multi class, single label problems, the two following algorithms were developed:

1. **AdaBoost.M1** was the first extension provided by Freund and Schapire. It is described to be the most direct algorithm, with only slight differences in comparison to the original AdaBoost algorithm. For the most part, the biggest difference was changing the binary algorithm's error rate from $|h_t(x_i) - y_i|$ to $[[h_t(x_i) \neq y_i]]$ "where, for any predicate π, we define $[[\pi]]$ to be 1 if π holds and 0 otherwise" [FS97]. This difference in the error is mainly concerned with the change from binary classifiers to multi class classifiers.

2. Following the AdaBoost.M1, the **AdaBoost.M2** algorithm was developed as a further solution to multi class problems. Unlike the AdaBoost.M1, which only varies slightly from the original AdaBoost algorithm, the AdaBoost.M2 needs an established communication line between the boosting algorithm itself and the weak PAC-learning algorithm. As Freund and Schapire describe it, this would "enable the weak learner to make useful contributions to the accuracy of the final hypothesis even when the weak hypothesis does not predict the correct label with probability greater than 1/2" [FS97].

In regards to multi class, multi label problems, the following algorithm was developed:
Linear regression and its analysis is a vital fragment of machine learning; helping us understand how dependent and independent variables interact within one another is key for a multitude of areas in scientific research. The AdaBoost.R algorithm, as set out by Freund and Schapire, aids in the boosting of regression problems. This algorithm differs from the original AdaBoost in its implementation of the distribution, p^t, as well as the calculation of the loss, h_t, and finally the finally hypothesis.

3 Applications Of The AdaBoost Algorithm

3.1 Basic Application of the AdaBoost Algorithm Using Decision Stumps

Decision stumps, *i.e.* one-level decision trees, are able to showcase the AdaBoost algorithm at its most basic applicable use. This is one of the reasons why decisions stumps are regularly used to aid in the explanation of the AdaBoost algorithm's application. Alongside that reason, no knowledge beyond the fundamentals revolving around decision trees is needed in order to be able to understand how this specific application works, thus making it the perfect way to communicate precisely how each step of the algorithm would be applied.

In this section, I will shortly explain what the definition of a decision stump is. Following that, I will promptly showcase how the decision stumps generated from data sets could be boosted using the AdaBoost to produce more accurate classifiers and a final hypothesis with an arbitrarily high accuracy.

3.1.1 Defining One-Level Decision Trees for AdaBoost Application

One-level decision trees, also known as decision stumps, form a basis for the basic application of the AdaBoost algorithm. The decision stump is made up of two main fragments, namely the root and the leaves [SW10]. Decision stumps do not necessarily need to be binary; they could essentially include all the possible values stemming from a specific function. For the original AdaBoost algorithm, however, binary decision stumps are used to showcase the algorithm's application. Variants of the AdaBoost algorithm, such as AdaBoost.M1 or AdaBoost.MH, would then be showcased on other types of decision stumps. An example of a binary decision stump can be seen in Fig. 3.

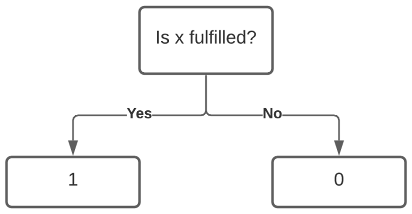

Figure 3: A basic decision stump with an input questioning whether the requirement "x" is fulfilled, and an output of either 1 or 0. Here, the root would be considered to be the question "is x fulfilled?" The leaves would then be the values 1 and 0.

Decision stumps, although only able to make decisions based on a single property, tend to also be surprisingly accurate. They are, reportedly, almost as accurate as full decision trees [Hol93]. This combination of relatively high accuracy, as well as being a weak enough classifier, would then enable us to use decision stumps as a possible algorithm behind what is described as a "**WeakLearn**" algorithm.

3.1.2 Implementation of the AdaBoost Algorithm on Decision stumps

Consider the following sample data set:

Table 1: Sample arbitrary data set.

Weight (Kg)	City Dweller	Exercises Regularly	Heart Disease
138	No	No	Yes
79	No	No	No
85	Yes	Yes	No
78	Yes	No	Yes
109	Yes	No	Yes
91	Yes	Yes	No

In Table 1, I chose to showcase a data set which contains three independent values: weight of the person, whether they live in the city and whether they exercise regularly. The fourth measurement shown in the table is the prediction. Put simply, we are using the first three measurements to predict the fourth one shown in the table.

Initially, each sample is given the same weight, which is equivalent to $\frac{1}{6}$, or approximately 0.167. Following this, a decision stump is created using the variable which best predicts the outcome[3]. This could be done by creating decision stumps for each of the variable, weight, city dwelling and regular exercise, and then seeing which one guessed the outcome correctly most times. This would produce the following three decision stumps:

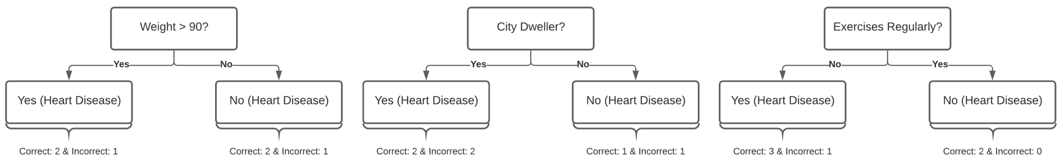

Figure 4: Sample stumps created from the variables in Table 1.

Here, as seen in Fig. 4, the variable of whether a person **exercises regularly** seems to have determined the outcome the most correctly. Therefore, we will choose this decision stump to be the first stump in our forest of decision stumps. The **total error** made by this stump would then be equivalent to the number of errors made multiplied by their weights, meaning that it would be $1 \cdot \frac{1}{6} = \frac{1}{6}$. This error would then be further utilised to determine **how much of a say that this specific decision stump would have in the decision trump forest**. Such a say could be determined using the following equation:

$$\text{Say within Forest} = \frac{1}{2}\log\left(\frac{1 - \text{Total Error}}{\text{Total Error}}\right) \tag{6}$$

In this specific example, the say for our initial decision stump would be about 0.805. Similarly, a **decision stump for the weight** would have had a total error of $\frac{1}{3}$ and a say of approximately 0.347, while a **decision stump for "city dwelling"** would have had a total error of $\frac{1}{2}$ and a say of 0. Now, with that done, each incorrectly classified sample will have its weight increased, while the rest will have their weights decreased. This would done using the following formula:

$$\text{New weight} = \text{current weight} \cdot e^{\text{say within the forest}} \tag{7}$$

Using this formula, the incorrectly identified error made by the decision stump for regular exercise would have a new weight of about 0.373. Likewise to Eq. 7, the new weight would have the exact same formula except with the say having a negative sign, giving us $e^{-(\text{say within the forest})}$. Using this updated equation, we get a new weight of 0.0745 for each of the correct samples. If we add up the weights[4], we get a **total weight** of 0.746, meaning that the **weights need to be normalised** in order to add up to 1. This is done by dividing each of the weights by the new sample total weight, such as follows:

$$\text{For each of the correctly identified samples: } \frac{0.0745}{0.746} \approx 0.0999. \tag{8}$$

$$\text{For the incorrectly identified sample: } \frac{0.373}{0.746} \approx 0.500. \tag{9}$$

Using these new, normalised weights, we can fundamentally produce a new set of samples that would focus more on the incorrectly identified sample. This process is then applied once again to the new set of samples. Using all of the stumps created through each further iteration, we will then sum the stumps that

[3]Here, as all samples have the same weight, one could essentially ignore the weights for the initial step.

[4]This would be done using the equation $5 \cdot 0.0745 + 1 \cdot 0.373$.

have classified the incorrectly classified sample, with the stumps that have a higher sum having the final say.

This sums up the first example of an application of the AdaBoost using simple decision stumps. As showcased, the algorithm attempts to adapt itself as each iteration goes on, with more attention given to particular samples through the variation of their weights.

3.2 Pedestrian Detection Through Motion And Appearance Patterns

A further application for the AdaBoost algorithm could be seen in the research done by Viola, Jones and Snow in their 2003 paper revolving around the detection of pedestrians. In their application, the motion detector is trained using the AdaBoost algorithm, with two specific variables: motion and appearance [VJS03].

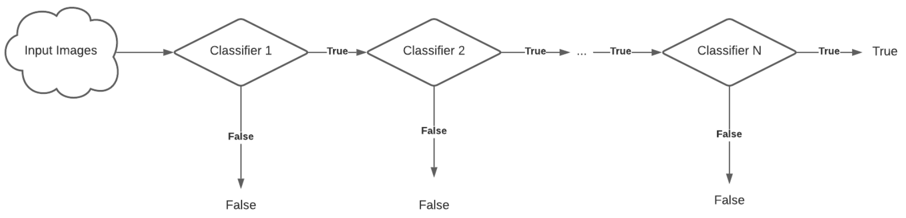

Figure 5: Architecture of a detection cascade showing the process in which an input is decided to be either true or false.

In Fig. 5, the cascade architecture for the training process of detector is shown. At each step, the classifier votes whether the input is true, in which case the classification continues, or whether it is false, in which the classification halts and the output is false. If an input is to be classified as true, it must go through a vote of all the classifiers first. This sort of process is very similar to the one described in Sec. 3.1.2. Simultaneously, the filters and optimal threshold for each round is chosen. With that, it is described that, "the resulting classifier balances intensity and motion information in order to maximize detection rates." This ties up to the aforementioned main use of the AdaBoost algorithm, which is the creation of an arbitrarily strong classifier using the given weak classifiers.

Figure 6: A sample provided of positive training images, using the training process described for Fig. 5. Image from [VJS03].

As stated in "Detecting pedestrians using patterns of motion and appearance", the AdaBoost algorithm used in each stage of Fig. 5 "reduces both the false positive rate and the detection rate of the previous stage". Using this sort of training process, a sample of positive images is produced, such as the one seen in Fig. 6.

Although the algorithm and process for pedestrian detection is significantly more complex than the one described in Sec. 3.1.2, it is also clear that the same simple process fundamentally applies to both decision stumps and pedestrian detection. Through its effectiveness, the AdaBoost algorithm is able to aid in the production of more solid positive samples, thus bolstering the whole process.

3.3 Face Detection

One other applicable research related to the AdaBoost algorithm is the visual object detection algorithm developed by Viola and Jones in 2001. This process utilises a learning algorithm based on the AdaBoost, which is given an amount of certain features in order to create "extremely efficient classifiers" [VJ01]. Furthermore, their algorithm utilises a detection cascades very similar to the one used in Fig. 5, as seen in Sec. 3.2.

In Sec. 2.2 of "Rapid Object Detection Using a Boosted Cascade of Simple Features", Viola and James describe how the AdaBoost algorithm is used for more than just one purpose. In developing their object detection, they made use of the AdaBoost in order to both train the classifiers and select the certain features or samples [VJ01], as previously mentioned. Picking the small set of features was mainly due to a concern regarding computing expenses, not necessarily due to the inefficiency os using a larger set of samples. Their "**WeakLearn**", or weak PAC-learning algorithm, selected the optimal rectangle that would be the most discernible when it comes to separating the positive and negative samples [VJ01]

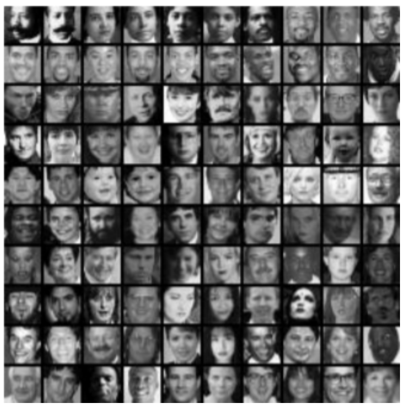

Figure 7: An example of a sample of training images used in [VJ01] to help train the face detection algorithm. Image from [VJ01].

In Fig. 7, we can see a set of images used for the AdaBoost-based training algorithm. Developing their 38-step, 6000-feature cascade helped Viola and James create an efficient, albeit rigorous, detector, with their detector being "about 15 times faster than an implementation of the detection system constructed by Rowley et al." [VJ01]

4 Further Related Work

The AdaBoost's importance in the field of boosting and machine learning does not only lay in its significance as an algorithm, but also the latter research conducted and algorithms created that were based on it. In order to obtain a better understanding of the AdaBoost's current applications and its importance, we will look at a few research papers that have built on the idea of boosting in general and the AdaBoost in specific. Some of these research papers have added onto the original AdaBoost algorithm, or one of its variants, in order to create a more efficient or more effective algorithm. This stands in terms of either the reduction of computation, noise or "confidence assigned to predictions" [SS99].

A multitude of the algorithms showcased in this paper were developed by either Freund, Schapire or both of them together. This also includes the further related pieces of research mentioned below. Nevertheless, there does exist related work developed by other researchers in terms of boosting. Examples of that would be the LPBoost, which A. Demiriz, K.P. Bennett and J. Shawe-Taylor worked on [DST02], or the CoBoosting algorithm, which Michael Collins and Yoram Singer developed [CS99].

4.1 LogitBoost

In LogitBoost, logistic impression is combined with the AdaBoost in order to minimise the error or the "logistic loss" [FHT00]. Unlike the original AdaBoost, the LogitBoost could be used with both binary and multi class problems.

4.2 Gentle AdaBoost

Gentle AdaBoost applies a different method to the minimisation of error. Whereas most algorithms would minimise the error "greedily", the Gentle AdaBoost chooses a specific coefficient to decrease the error with [SS99].

4.3 BrownBoost

The BrownBoost, an algorithm put forward by Yoav Freund which expands on his own "boost by majority algorithm", usually performs better with noisy datasets, *i.e.*sets of data with an array of other useless information, which is the noise [Fre01]. Incidentally, this sort of algorithm is not completely similar to the AdaBoost. That is due to its utilisation of a "non-convex loss function", thus enabling it to circumvent most noisy data sets.

4.4 Comparison of the AdaBoost, LogitBoost and BrownBoost's Effectiveness

As previously mentioned, the LogitBoost and BrownBoost differ from the original AdaBoost algorithm in different manners; with the LogitBoost focusing on minimising the error and the BrownBoost focusing on enabling the boosting of noisy data sets. In Table 2, we can see the error rate for each of the algorithms given a specific data set that has had artificial noise built into it.

Table 2: Error rates for the AdaBoost, LogitBoost and BrownBoost on data sets with 20% artificial noise. Table from [ME03].

	AdaBoost		LogitBoost		BrownBoost	
Data set	Training Error	Test Error	Training Error	Test Error	Training Error	Test Error
Wisconsin	0.216 ± 0.003	0.238 ± 0.005	0.190 ± 0.004	0.238 ± 0.009	0.188 ± 0.001	0.230 ± 0.004
Credit	0.239 ± 0.003	0.316 ± 0.010	0.190 ± 0.005	0.281 ± 0.009	0.177 ± 0.002	0.289 ± 0.005
Wine	0.088 ± 0.003	0.287 ± 0.008	0.171 ± 0.012	0.256 ± 0.019	0.165 ± 0.003	0.255 ± 0.011
Balance	0.214 ± 0.003	0.337 ± 0.006	0.185 ± 0.005	0.247 ± 0.012	0.158 ± 0.002	0.280 ± 0.006

From the table, it is clear that the BrownBoost algorithm is overall the best algorithm for training; averaging a training error of 0.172 ± 0.002, while LogitBoost and AdaBoost training errors average at 184 ± 0.0065 and 189.25 ± 0.003 respectively. Those coinsides with the aforementioned main purpose of the BrownBoost, namely its ability to deal with noisy data sets. The same applies for test errors, with BrownBoost having the lowest average error, followed by LogitBoost and finally AdaBoost.

5 Conclusion

At this point, it should be clear how significant the AdaBoost algorithm is in the field of machine learning; its applications have clear, significant uses for learning algorithms, thus providing us with a momentous tool to be used in tandem with other machine learning techniques.

Essentially, the AdaBoost has bolstered a then-new field of machine learning while answering the several questions laid out by Michael Kearns in 1988. This significant piece of research has opened up ability to create powerful classifiers with small data sets and relatively weaker classifiers. Applications for the AdaBoost do not necessarily stop at motion or face detection, but they could also be applied in the field of data science in order to help scientists with their work in structuring data and utilising it.

References

[CS99] Michael Collins and Yoram Singer. Unsupervised models for named entity classification. *researchgate:publication/2481707*, 1999.

[DST02] Bennett Kristian P. Demiriz, Ayhan and John Shawe-Taylor. Linear programming boosting via column generation. *doi:10.1023/A:1012470815092*, 2002.

[FHT00] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: A statistical view of boosting. *doi:10.1214/aos/1016218223*, 2000.

[Fre95] Yoav Freund. Boosting a weak learning algorithm by majority. *doi:10.1006/inco.1995.1136*, 1995.

[Fre01] Yoav Freund. An adaptive version of the boost by majority algorithm. *doi:10.1023/A:1010852229904*, 2001.

[FS97] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *jcss:1997.1504*, 1997.

[Hol93] Robert C. Holte. Very simple classification rules perform well on most commonly used datasets. *doi:10.1023/A:1022631118932*, 1993.

[Kea88] Michael Kearns. Thoughts on hypothesis boosting. *https://www.cis.upenn.edu/ mkearns/papers/boostnote.pdf*, 1988.

[ME03] Hand David J. McDonald, Ross A. and Idris A. Eckley. An empirical comparison of three boosting algorithms on real data sets with artificial class noise. *doi:10.1007/3-540-44938-8*$_4$, 2003.

[Sch90] Robert E. Schapire. The strength of weak learnability. *doi:10.1007/BF00116037*, 1990.

[SS99] Robert E. Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *doi:10.1023/A:1007614523901*, 1999.

[SW10] Claude Sammut and Geoffrey I. Webb. *Decision Stump*, pages 262–263. 2010.

[Val84] Leslie Valiant. A theory of the learnable. *doi:10.1145/1968.1972*, 1984.

[VJ01] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. *doi:10.1109/CVPR.2001.990517*, 2001.

[VJS03] Paul Viola, Michael Jones, and D Snow. Detecting pedestrians using patterns of motion and appearance. *doi:10.1109/ICCV.2003.1238422*, 2003.