LinearBoost for Classification

by

Dekai Lin

B.Sc., Simon Fraser University, 2020

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

in the
School of Computing Science
Faculty of Applied Sciences

Declaration of Committee

Name:	**Dekai Lin**
Degree:	**Master of Science**
Thesis title:	**LinearBoost for Classification**
Committee:	**Chair:** Steven Bergner Lecturer, Computing Science
	Ke Wang Supervisor Professor, Computing Science
	Maxwell Libbrecht Committee Member Assistant Professor, Computing Science
	Sharan Vaswani Examiner Assistant Professor, Computing Science

Abstract

AdaBoost is a well-studied and widely used ensemble method that improves classification performance iteratively by focusing on misclassified samples through reweighting. It assigns higher weights to misclassified samples in each training iteration and makes the final predictions through voting of all base classifiers learned from all iterations. This reweighting, however, can lead to a disproportionate focus on properly classified samples in early iterations, resulting in many inferior classifiers for such samples in the final voting. In this work, we propose *LinearBoost*, a competing ensemble approach, to address this issue. Instead of voting by multiple classifiers, LinearBoost classifies a sample by the first "promising" classifier learned in the iterative process, where "promising" means a prediction having a high enough confidence of being correct. The next iteration of training will focus on the remaining samples that do not have a promising prediction. Therefore, LinearBoost can maintain the performance of properly classified samples in early iterations (by identifying their promising classifiers) while improving the performance of misclassified samples iteratively. LinearBoost is a general boosting strategy that can work with any type of base classifiers. Experiments on datasets with different characteristics and different types of base classifiers show that LinearBoost usually does better than AdaBoost and its variations, achieving higher Macro and Weighted F1 scores.

Keywords: Ensemble; Boosting; AdaBoost

Dedication

For my parents.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my supervisor, Dr. Ke Wang, for his unwavering guidance, invaluable feedback, and continuous encouragement throughout the course of this research. His expertise and insights have been instrumental in shaping this work, and I am truly fortunate to have had him as my mentor.

I am also immensely grateful to Dr. Maxwell Libbrecht, who as a committee member, provided critical feedback and suggestions that greatly enhanced the quality of this thesis. His keen observations and constructive critiques were invaluable.

Special thanks to Dr. Sharan Vaswani for his role as the examiner. His thorough review and insightful comments not only improved this thesis but also broadened my perspective on the subject.

I would like to extend my appreciation to Dr. Steven Bergner, the chair, for his words of encouragement were particularly uplifting. After the defense, when I was grappling with self-doubt, it was Dr. Bergner's reassurance and positive feedback that restored my confidence and spirit. His kindness and support during that pivotal moment will always be remembered.

On a personal note, I owe a debt of gratitude to my parents, whose unwavering support and belief in me have been the bedrock upon which I built my academic journey. Their sacrifices, love, and encouragement have been my guiding light, pushing me to strive for excellence.

Last but certainly not least, I want to thank my friends, who stood by me through the highs and lows of this journey. Their constant encouragement and understanding made the challenging times bearable and the good times even more memorable.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Classification is the task of assigning predetermined classes to data samples based on their features. In real-world classification tasks, classifiers are not always successful across all classes in a dataset. Class competition, i.e., numerous classes that are readily misclassified from one another, such as shirt and T-shirt picture classification, may explain poor performance in certain classes [24]. Another factor is an imbalanced-class distribution, which causes model training to favor majority classes with a higher share of data, resulting in poor minority class performance [38]. Improving the performance of under-performed classes is critical for obtaining a generally well-performed classifier.

Ensemble methods achieve accurate results by combining the predictions of multiple base classifiers, which leverages these base classifiers' collective knowledge and diversity [28]. AdaBoost [14], which sequentially trains a group of base classifiers in multiple iterations, is one of the most researched ensemble methods. AdaBoost trains a base classifier in each iteration and utilizes the base classifier's error rate to raise the weight of misclassified samples in the following iteration. The final prediction for each sample is determined by a weighted majority voting from all base classifiers. AdaBoost is a general boosting methodology that can be applied to different kinds of base classifiers, such as support vector machine, decision tree, and neural network, to outperform a single base classifier's performance [23, 15, 32, 37].

AdaBoost boosts the performance of under-performed samples by paying more attention (i.e., weights) to misclassified samples in the next iteration. However, this inevitably hurts the samples that were correctly classified in the current iteration because their weights will be reduced relatively in the next iteration. Consequently, the final predictions for such samples is determined by the vote of many weaker base classifiers (due to reduced weights). The principle underlying AdaBoost is that aggregating several weak learners (i.e., performing slightly better than random guessing) can yield a final learner that has a better performance than any of the input weak learners used for the aggregation [31]. However, this does not answer the question of whether this aggregation outperforms the alternative approach of choosing a "strong" base classifier *individually* for each sample. Instead of involving all learned weak base classifiers in the final voting, the latter approach "locks in" a single, usually strong, base classifier for each sample. This thesis aims to investigate this competing ensemble approach.

Our observation is that if a sample has a high likelihood of being correctly classified by the base classifier in the current iteration, this prediction should be reserved as the *final* prediction for the sample, rather than the voting by a number of weaker base classifiers learned from small weights given to this sample. Based on this observation, we propose ***LinearBoost***, an ensemble method that, like AdaBoost, iteratively learns a sequence of base classifiers, but unlike AdaBoost, and classifies a sample using the first base classifier that has a "confident prediction" for that sample. See more details below.

At each iteration, the proposed method identifies and removes the samples that are classified by the current base classifier with high confidence. Such samples are removed as subsequent iterations do not need to make predictions for such samples (because they have already been classified by the current base classifier), and subsequent iterations should focus on the remaining samples that are difficult to classify.

The objective of the thesis: The objective of this thesis is to study the proposed solo confident base classifier for each samples, i.e., LinearBoost, vs AdaBoost's aggregation of all base classifiers. Like AdaBoost, LinearBoost can be applied to any learning algorithm Alg for base classifiers by calling Alg as a black-box during training. This property allows LinearBoost to be used to boost the performance of any existing Alg. In particular, let $LinearBoost(Alg, Data, M)$ and $AdaBoost(Alg, Data, M)$ denote the resulting ensemble classifiers learned by applying LinearBoost and AdaBoost to training data $Data$, respectively, where Alg is the learning algorithm applied at each iteration to generate a base classifier and M is the number of iterations. Our main claim is that $LinearBoost(Alg, Data, M)$ can get a better performance than Alg and $AdaBoost(Alg, Data, M)$ for different $Data$ and Alg in terms of generalization.

The contributions of this thesis are as follows: (1) We propose LinearBoost to address the reweighting of samples in AdaBoost that creates weaker base classifiers for samples that were already well-classified. LinearBoost is a general boosting strategy that, similar to AdaBoost, can be applied to any learning algorithm Alg for learning the base classifier in each iteration. (2) To implement LinearBoost, we address several technical concerns, including the formalization of "confident prediction", promoting confident prediction, and over-fitting caused by decreased training size in later iterations. (3) We compare the efficacy of LinearBoost vs AdaBoost and its variants on datasets with varying features and discuss our findings. The study demonstrates that LinearBoost can sustain the performance of well-performed samples while improving the performance of under-performed samples. In particular, LinearBoost improves the single base classifier Macro F1 by 2.8% and 9.6% on average for balanced-class and imbalanced-class datasets, respectively, compared to 0.4% and 5.1% improvements by the best-performing AdaBoost's variants.

The structure of the thesis is organized as follows: Chapter 2 introduces AdaBoost algorithm, AdaBoost's variants and other ensemble methods, as well as the evaluation metrics of classifiers. Chapter 3 presents the technical details of LinearBoost. Chapter 4 considers the parameter settings of LinearBoost and analyzes time complexity. Chapter 5 reports the experimental evaluation of LinearBoost and our analysis. Finally, we conclude the thesis.

Chapter 2

Related Work

For comparison purposes, we present several ensemble methods in this chapter, including the original AdaBoost and its variants, as well as the evaluation metric in this chapter.

2.1 AdaBoost

The original AdaBoost proposed by [14] was for binary-class classification tasks. Algorithm 1 (SAMME) is the generalization for K-class classification tasks from [18]. When $K = 2$, Algorithm 1 performs the same as the original AdaBoost. AdaBoost's core principle is to train a base classifier using current sample weights w and then increase the weights of misclassified samples in the following iterations. AdaBoost progressively trains M base classifiers on the whole training set S and adjusts the sample weights w in each iteration. In the first iteration, AdaBoost uniformly initializes the sample weights, i.e., $w_i = \frac{1}{n}$ for all samples, $i = 1, \ldots, n$, where n is the total number of training samples.

At iteration m, the base classifier $T^{(m)}$ is fitted to minimize a weighted loss function based on the current sample weights w (Equation 2.1):

$$\min \sum_{i=1}^{n} w_i \cdot Loss(y_i, T^{(m)}(x_i)) \tag{2.1}$$

where y_i and $T^{(m)}(x_i)$ are the true and predicted class for the sample x_i respectively, and $Loss(\cdot, \cdot)$ is the loss function that measures the difference between the true class and the predicted class. Line 4 calculates the error rate $\epsilon^{(m)}$ of the current base classifier $T^{(m)}$, where $\mathbb{1}$ is the indicator function, and Line 5 computes the base classifier $T^{(m)}$'s weight $\alpha^{(m)}$ based on the error rate. A base classifier $T^{(m)}$ with a low error rate $\epsilon^{(m)}$ will get a large classifier weight $\alpha^{(m)}$, which is used in Line 6 for updating the sample weights w_i of all training samples. Importantly, a larger $\alpha^{(m)}$ leads to a larger weight for the base classifier $T^{(m)}$ and the misclassified samples x_i. The normalized weights w (Line 7) are used in the next base classifier training iteration. The final ensemble classifier is given by Line 9 as the weighted aggregation of $T^{(m)}$, for $m = 1, \cdots, M$, with $\alpha^{(m)}$ as the weights.

Algorithm 1 AdaBoost-SAMME [18]

```
Input: Training set $S = \{(x_i, y_i)\}$, $i = 1, \ldots, n$; and $y_i \in \mathbb{C}$, a set of classes $\mathbb{C} = \{1, \ldots, K\}$;
$M$: number of iterations; $T$: model.
Output: Predicted classes $H(x)$.
1: Initialize the training set $S$ weights $w_i = \frac{1}{n}, i = 1, \ldots, n$.
2: for $m = 1$ to $M$ do
3:     Fit a base classifier $T^{(m)}(x)$ with the training set $S$ using weights $w$.
4:     $\epsilon^{(m)} = \sum_{i=1}^{n} w_i \cdot \mathbb{1}\left(y_i \neq T^{(m)}(x_i)\right)$.
5:     $\alpha^{(m)} = \log \frac{1-\epsilon^{(m)}}{\epsilon^{(m)}} + \log(K-1)$.
6:     $w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{1}\left(y_i \neq T^{(m)}(x_i)\right)\right)$.
7:     Re-normalize $w$.
8: end for
9: return $H(x) = \underset{k}{\operatorname{argmax}} \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{1}(T^{(m)}(x) = k)$.
```

AdaBoost has the convergence property as stated in Theorem 1.

Theorem 1 ([3])**.** *Let D be a distribution over $X \times \{-1, 1\}$, and let S be a training set of n samples chosen independently at random according to D. Suppose the base learning algorithm, when called by AdaBoost f, generates base classifiers with weighted training errors $\epsilon^{(1)} \ldots \epsilon^{(M)}$. Then for any θ, we have a probability on S that:*

$$P_S[yf(x) \leq \theta] \leq 2^M \prod_{m=1}^{M} \sqrt{\left(\epsilon^{(m)}\right)^{1-\theta} \left(1-\epsilon^{(m)}\right)^{1+\theta}} \tag{2.2}$$

$yf(x)$ is the margin of the sample x with the true class y. For binary-class classification, y is from $\{-1, 1\}$ and $f(x)$ is the predicted value in the range $[-1, 1]$, so $yf(x) \leq 0$ represents an incorrect prediction of the sample x where the predicted value $f(x)$ and the true class y have the opposite signs. The theorem implies that if $\epsilon^{(m)} \leq \frac{1}{2} - \gamma$ for some $\gamma > 0$ and $\theta < \gamma$ in all M iterations (i.e., the base classifier in every iteration performs better than the binary-class random guessing), $P_S[yf(x) \leq \theta]$ will decrease and approach 0 as M increases.

Theorem 2 gives a bound on the generalization error of AdaBoost when the task is binary-class classification (the generalization error theorems for K-class classification is in Theorem 6 of [3]):

Theorem 2 ([3])**.** *Suppose the base classifier space $\mathbb{H}$ has VC-dimension d, and let $\delta > 0$. Assume that $n \geq d \geq 1$. Then with the probability at least $1 - \delta$ over the random choice of the training set S, every weighted average function f satisfies the following bound for all $\theta > 0$:*

$$P_D[yf(x) \leq 0] \leq P_S[yf(x) \leq \theta] + O\left(\frac{1}{\sqrt{n}}\left(\frac{d \log^2(n/d)}{\theta^2} + \log(1/\delta)\right)^{1/2}\right) \tag{2.3}$$

According to Theorem 1, as M increases, the term $P_S[yf(x) \leq \theta]$ goes to 0 when $\epsilon^{(m)} \leq \frac{1}{2} - \gamma$ (with $\gamma > 0$ and $\theta < \gamma$) in all M iterations, and the second term $O(\cdot)$ doesn't depend on M. Therefore, the generalization error $P_D[yf(x) \leq 0]$ is bounded and decreases as M increases.

However, we need to take the above with a grain of salt. As pointed out by the authors of AdaBoost [3], this generalization bound is very loose and only until the training set size is in the tens of thousands then the bound becomes useful. Also, Theorem 2 assumes that the VC-dimension d of the base classifier space is not greater than the training sample size n, i.e., $n \geq d \geq 1$. This implies that Theorem 2 cannot be applied in many practical cases. For example, the VC-dimension of a neural network has the worst case $O(|E|^2 \cdot |V|^2)$, where $|E|$ is the number of edges and $|V|$ is the number of nodes in the network [39]. Take the image recognition deep CNN VGG16 as an example, where the last two fully connected hidden layers have 4,096 nodes each [34]. So for the last two hidden layers alone, $|V| = 8,192$ and $|E| = 4,096^2$. The VC-dimension in this case is a large number, i.e., $O(|E|^2 \cdot |V|^2)$, and Theorem 2 applies only for training data of size larger than this VC-dimension.

In addition, Theorem 2 considers a *given* base classifier space. It remains an open question of how to choose the base classifier space so that these bounds are tight. A complex base classifier space would lead to a large VC-dimension d, thus, a loose bound, on the other hand, a simple base classifier space would lead to under-fitting, i.e., $\epsilon^{(m)} \leq \frac{1}{2} - \gamma$ may not hold, as required for the convergence of binary-class classification. Thus, the choice of the base classifier space will affect the generalization, and how to choose a good base classifier space is not addressed by Theorem 2.

For the above reasons, Theorem 2 is more of a theoretical value than a practical one. In practice, the authors of AdaBoost suggest using a validation set to select the number of iterations, M, to reduce the generalization error [14].

2.2 Variants for Multi-class Data

SAMME. The original AdaBoost is for binary-class classification. Algorithm 1 SAMME is from [18], a multi-class variant of AdaBoost. SAMME uses the same structure as AdaBoost with a minor but subtle difference. Compared with the original AdaBoost, SAMME has an extra term $\log(K-1)$, where K is the number of classes (Algorithm 1 Line 5). This term ensures that $\alpha^{(m)}$ is positive as long as the base classifier is better than the K-class random guessing, i.e., $\epsilon^{(m)} < \frac{K-1}{K}$.

SAMME.R. SAMME.R is a variant of SAMME [46]. SAMME.R updates the sample weights using real-valued confidence-rated predictions, while SAMME just uses classifications. Compared with SAMME, SAMME.R method uses the base classifier's prediction confidence to update the weights, thus, it usually converges quicker and has a smaller test error.

2.3 Variants for Imbalanced-class Data

AdaBoost and its multi-class variants above prioritize misclassified samples to minimize the classification error [22]. In many imbalanced-class classifications, the focus is more on the prediction accuracy of some target classes. For example, an important binary-class classification aims to correctly classify the positive class (as the target class), and the positive class is usually the minority class, e.g., buyer (positive) vs non-buyer (negative), malware (positive) vs benign software (negative). In such problems, the classification should pay more attention to the False Negative samples as its potential cost of misclassification is high, i.e., misclassifying malware as benign software, or misclassifying a buyer as non-buyer. Three AdaBoost's variants have been designed for imbalanced-class distribution, all of which assign a higher *Cost* to misclassified minority class samples to increase their weights during training.

AdaCost. The idea of AdaCost is assigning higher costs *Cost* (range $[0, 1]$) to minority class samples and adjusting the sample weight during training to emphasize the minority class classification accuracy [13]. Specifically, during the weight updating, AdaCost increases the weights of misclassified minority class samples more than misclassified majority class samples. Furthermore, the weights of correctly classified minority class samples decrease less compared with the correctly classified majority class samples. Equation 2.4 can combine with the sample weights w updating to achieve the requirements mentioned above

$$\beta_i = \begin{cases} -0.5Cost_i + 0.5, & y_i = T^{(m)}(x_i) \\ \ \ 0.5Cost_i + 0.5, & y_i \neq T^{(m)}(x_i) \end{cases} \tag{2.4}$$

where $Cost_i$ is the cost term value of sample x_i and updating the sample weight in Algorithm 1 Line 6: $w_i \leftarrow w_i \cdot \exp\left(\beta_i \cdot \alpha^{(m)} \cdot \mathbb{1}\left(y_i \neq T^{(m)}(x_i)\right)\right)$.

Asymmetric AdaBoost. Asymmetric AdaBoost imposes various cost values depending on sample types, leading to asymmetrically updating to the weights of different misclassified samples, i.e., False Positive and False Negative samples update asymmetrically [40]. The cost setting of Asymmetric AdaBoost is shown in Equation 2.5

$$Cost_i = \begin{cases} \sqrt{G}, & y_i \text{ is positive class} \\ \dfrac{1}{\sqrt{G}}, & y_i \text{ is negative class} \end{cases} \tag{2.5}$$

where False Negative samples' cost value is G times more than False Positive samples'. The users can base on the actual cost of False Positive samples and False Negative samples in real-world applications to determine the value of G. The sample weight update formula in Asymmetric AdaBoost is: $w_i \leftarrow \exp(\frac{1}{M}\log(Cost_i)) \cdot w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{1}\left(y_i \neq T^{(m)}(x_i)\right)\right)$ where M is the number of iterations.

AdaC. Same as AdaCost, AdaC series adds cost terms into the learning procedure to increase the minority class impact [36]. The cost of each sample demonstrates the significance of accurately identifying that sample. It uses a higher cost for minority class samples and a lower cost for majority class samples, thus enhancing minority class sample weights. For example, the total number of samples divided by the number of samples in each class can be used as a cost term to achieve the above requirement (Equation 2.6)

$$Cost_k = \frac{n}{n_k}, \text{for } k = 1 \dots K \tag{2.6}$$

where n is the total number of samples and n_k is the number of samples in a class k. So a class with a smaller sample size has a larger cost term. AdaC has three variants, AdaC1, AdaC2 and AdaC3. The difference between them is how to add cost terms into the sample weight update, for example, AdaC2 introduces the cost term outside the exponential term: $w_i \leftarrow Cost_k \cdot w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{1}\left(y_i \neq T^{(m)}(x_i)\right)\right)$ where x_i's true class is k.

These algorithms use a framework similar to AdaBoost, with the distinction being how the cost term is included in the weighting scheme. It is worth noting that when the $Cost$ setting is the same for all x_i, these methods reduce to standard AdaBoost. All of these methods need the user to input the cost term, which is not straightforward in practice [44]. For example, what is the cost term for False Negative where the positive class represents malware software and the negative class represents benign software?

2.4 Other Ensemble Methods

The ensemble strategy of AdaBoost and its variants are based on sample reweighting to assign larger weights to misclassified samples. Other ensemble strategies exist as well. Below, we briefly review some popular ensemble methods.

Bagging. The goal of Bagging is to reduce the variance of a base classifier's predictions, i.e., reduce overfitting, by introducing randomness into the training process [5]. Compared with AdaBoost, Bagging's ensemble strategy assigns the same weights to all training samples and votes the predictions with the majority from the multiple base classifiers that have been trained on different subsets of the training samples, while AdaBoost iteratively adjusts the weights of training samples to train each base classifier with the same set of samples and votes their predictions weighted on the base classifiers' performance. The training samples for each base classifier in Bagging are randomly selected from the whole training set with replacements. This training sample randomness introduces variation in the training process and helps Bagging to learn more robust and generalized patterns.

Random Forest. Random Forest builds upon Bagging by introducing additional feature randomness in the training process [6]. Feature randomness can reduce overfitting by reducing the correlation between decision trees in the forest, as the whole set of features may contain some correlated features which may provide duplicate information and produce trees with high correlation. Using a random subset of features for each tree can break up the correlation between the features and create more diverse trees.

GradientBoost. Gradient Boosting combines several base classifiers, usually decision trees, to form an ensemble classifier by progressively minimizing the residual of the ensemble classifier in each iteration [17]. Residual refers to the difference between the actual observed value and the predicted value of a sample. For example, in a classification task, if the true class' predicted probability value of a sample is 70%, then the residual of this sample is 30% as the actual observed probability value of the true class is 100%. Instead of training the base classifier on the sample observed value like AdaBoost, the base classifier generated by GradientBoost is trained on the residual of the ensemble classifier which is the combination of all base classifiers learned in previous iterations. Therefore, the base classifier can predict the residual of the ensemble classifier so that adding this base classifier into the ensemble classifier can make a more accurate prediction because it knows how far away is the observed value via residual.

In order to minimize the residual of the ensemble classifier, GradientBoost uses a loss function to measure the residual and minimizes this loss function by updating the predicted values in the direction of the negative gradient of the loss function. Negative gradient points to the steepest descent of the loss function, so updating the predicted values in this direction can minimize the loss function values, i.e., residual. This learning process is repeated for each base classifier in GradientBoost, therefore, the predicted value is iteratively updated in the direction which can minimize the residual and it gradually approaches the observed value through multiple base classifiers.

XGBoost. XGBoost is an advanced implementation of GradientBoost and it is one of the state-of-the-art ensemble methods, with mitigating overfitting, split finding and handling missing values in training [8]. Overfitting is handled by adding a regularization term in the training and the split finding is based on approximation. Also, XGBoost is able to handle missing values as the default path for missing values is constructed during the tree building.

There are many other ensemble methods that are not covered in this section, e.g., LogitBoost [16], SMOTEBoost [7], LightGBM [20]. As mentioned in Introduction, the objective of this thesis is to study the effectiveness of the single base classifier's prediction of LinearBoost vs the aggregation prediction of AdaBoost. Because the boosting strategies of the ensemble methods mentioned in this section are different from AdaBoost's, our comparison focuses on LinearBoost and AdaBoost only, i.e., $LinearBoost(Alg, Data, M)$ vs $AdaBoost(Alg, Data, M)$ with the same Alg, $Data$ and M.

However, since LinearBoost can be applied to any existing learning algorithm Alg through $LinearBoost(Alg, Data, M)$, Alg can be any of the above ensemble methods. If our objective that LinearBoost boosts the performance of any Alg can be achieved, i.e., $LinearBoost(Alg, Data, M)$ has a better performance than $Alg(Data, M)$, our work also improves these ensemble methods' performance and this is our way of LinearBoost surpassing the state-of-the-art ensemble methods.

2.5 Evaluation Metrics

To assess model performance, we use Macro Average F1 score (**Macro F1**) and Weighted Average F1 score (**Weighted F1**). In order to get these two metrics values, the following three measures for each class k need to be collected:

True Positive (TP_k): a collection of samples whose true classes are k and correctly predicted as class k. In this case, class k is treated as the positive class. Same below.

False Positive (FP_k): a collection of samples whose true classes are not k and incorrectly predicted as class k.

False Negative (FN_k): a collection of samples whose true classes are k and incorrectly predicted as other classes.

Then, we define Precision, Recall and F1 score for each class k as following:

Precision ($Precision_k$): the proportion of samples whose true classes are k among the samples predicted as class k (Equation 2.7).

$$Precision_k = \frac{TP_k}{TP_k + FP_k} \tag{2.7}$$

Recall ($Recall_k$): the proportion of samples predicted as class k among the samples whose true classes are k (Equation 2.8).

$$Recall_k = \frac{TP_k}{TP_k + FN_k} \tag{2.8}$$

F1 ($F1_k$): the harmonic mean of precision and recall for class k (Equation 2.9).

$$F1_k = 2\frac{Precision_k \cdot Recall_k}{Precision_k + Recall_k} \tag{2.9}$$

Finally, Macro F1 and Weighted F1 for a set of samples are provided by:

$$Macro\,F1 = \frac{\sum_{k=1}^{K} F1_k}{K} \tag{2.10}$$

where K is the total number of classes.

$$Weighted\,F1 = \sum_{k=1}^{K} \frac{n_k}{n} \times F1_k \tag{2.11}$$

where n_k is the number of samples in a class k and n is the total number of samples. In other words, "Macro" is the simple average over all classes and "Weighted" is the weighted average over all classes. Similarly, we can define Macro Precision.

The difference between Macro F1 and Weighted F1 is minor for a balanced-class distribution (i.e., n_k is comparable for all classes k). When the class distribution is skewed, Macro F1 is generally less than Weighted F1 because a larger class usually has a higher F1 score than a smaller class. The key evaluation metric in this thesis is **Macro F1** as it is more robust than accuracy (the percentage of samples correctly classified) and Weighted F1 because all classes are weighted equally.

Chapter 3

Main Method

This chapter introduces the algorithms for LinearBoost, leaving the parameter settings and time complexity analysis to the next chapter.

3.1 Training Phase

AdaBoost reweights all samples in each training iteration such that correctly classified samples have smaller weights and incorrectly classified samples have larger weights. In other words, correctly classified samples get less attention in the next training iteration, which results in a weaker base classifier $T^{(m)}$ in the next iteration for such samples in the final majority vote. As discussed in Section 2.1, the theoretical bound on the generalization error of AdaBoost holds only for a large training data set and it is too loose to be practically useful. Furthermore, those bounds assume a given base classifier space, but how to choose a base classifier space to have a tight bound is left unaddressed.

In this section, we present LinearBoost as an alternative to AdaBoost. Our observation is that if a sample has a high likelihood of being correctly predicted, i.e., a **confident prediction**, we should *lock in* the prediction as the *final* prediction for the sample and not consider the sample in the next iteration of training. In this way, each sample is classified by the first base classifier that can provide a confident prediction, and subsequent iterations concentrate on the remaining samples that do not have confident predictions.

Firstly, we formalize the notion of "confident prediction". For a threshold `TH` and a given validation set V, if a class k's precision, $Precision_k$, is at least `TH` on V, i.e., the portion of true class k among those predicted as class k is at least `TH` (Equation 2.7), we consider that the samples predicted as class k have *confident predictions*, with respect to `TH`. In this case, we have at least `TH` confidence that the samples predicted as class k are correctly predicted. If V is representative of unseen samples, we expect that this prediction confidence will hold on unseen samples as well. While high precision of class k is enforced by confident prediction, the recall of class k is improved iteratively by picking up the remaining samples of class k over multiple iterations.

In light of the above, we propose LinearBoost in Algorithm 2. LinearBoost has a structure similar to the multi-class AdaBoost, i.e., SAMME, although it varies from AdaBoost in numerous significant aspects, as discussed below.

Algorithm 2 LinearBoost – Training phase

Input: Training set $S' = \{(x_i,\ y_i)\}$, $i = 1, \ldots, n_{s'}$; Validation set $V = \{(x_i,\ y_i)\}$, $i = 1, \ldots, n_v$; and $y_i \in \mathbb{C}, \mathbb{C} = \{1, \ldots, K\}$; M: number of iterations; T: model; `TH`: precision threshold.

Output: A list of M base classifiers.

```
1: Initialize the training and validation weights $w$ separately and uniformly.
2: for $m = 1$ to $M$ do
3:     Fit a base classifier $T^{(m)}(x)$ with the training set $S'$ using weights $w$.
4:     for class $k \in \mathbb{C}$ do
5:         if $Precision_k(V) \geq$ TH then
6:             Add class $k$ into confident classes list $C^{(m)}$.
7:         end if
8:     end for
9:     for class $k \notin C^{(m)}$ do
10:        $\epsilon_k^{(m)} = \frac{\sum_{(x_i,y_i)\in V} w_i \cdot \mathbb{1}\left(y_i \neq T^{(m)}(x_i)\right) \cdot \mathbb{1}\left(T^{(m)}(x_i)=k\right)}{\sum_{(x_i,y_i)\in V} w_i \cdot \mathbb{1}\left(T^{(m)}(x_i)=k\right)}$.
11:        $\alpha_k^{(m)} = \log \frac{1-\epsilon_k^{(m)}}{\epsilon_k^{(m)}} + \log(K-1)$.
12:    end for
13:    for $\forall(x_i, y_i) \in S' \cup V$ do
14:        if $T^{(m)}(x_i) = k \in C^{(m)}$ then
15:            Remove $x_i$ from the corresponding $S'$ or $V$.
16:        else
17:            $w_i \leftarrow w_i \cdot exp\left(\alpha_k^{(m)} \cdot \mathbb{1}\left(y_i \neq T^{(m)}(x_i)\right)\right)$.
18:        end if
19:    end for
20:    if $S' = \emptyset$ or $V = \emptyset$ then
21:        Set $\alpha^{(m+1)} \ldots \alpha^{(M)} = \mathbf{0}$.
22:        Exit the for loop of Line 2.
23:    end if
24:    Re-normalize training and validation weights $w$ separately.
25: end for
26: return $L = \{(T^{(1)}, C^{(1)}, \alpha^{(1)}), \ldots, (T^{(M)}, C^{(M)}, \alpha^{(M)})\}$.
```

Input. Compared with Algorithm 1, LinearBoost has two more inputs, a validation set V and a precision threshold `TH`, which are used to identify confident prediction as we mentioned above. We will discuss the choice of `TH` in Section 4.2. In LinearBoost training, we divide the original n samples training set S into two disjoint sets. One set contains $n_{s'}$ samples and we use it as the training set S'. The other set contains n_v samples and we use it as the validation set V $(n = n_{s'} + n_v)$.

Determine confident prediction (Line 4-8). LinearBoost diverges significantly from AdaBoost in that, in each iteration m, we find the classes that have *confident predictions* by the current base classifier $T^{(m)}$ and use $T^{(m)}$ to predict such classes. These classes, represented by $C^{(m)}$, are the classes k having $Precision_k \geq \texttt{TH}$, calculated on the current iteration validation set V. In other words, if a sample is predicted to have a class k from $C^{(m)}$, the probability that k is the true class is at least TH.

Compute per-class weights (Line 9-12). Our weighting strategy is designed to push the classes $k \notin C^{(m)}$ into $C^{(m+1)}$ in the next iteration. This is done by increasing $Precision_k$ to pass TH in the next iteration. In order to maximize the number of such classes, we assign more weights to the classes $k \notin C^{(m)}$ that have a higher precision (i.e., closer to TH) because it is easier for such classes to pass TH. With this in mind, Line 10 computes the class-specific error rate $\epsilon_k^{(m)}$ for each predicted class $k \notin C^{(m)}$. Note that we use the indicator function $\mathbb{1}\left(T^{(m)}(x_i) = k\right)$ to ensure that the error rate is calculated only using the samples whose predicted classes are k. A class $k \notin C^{(m)}$ having a precision closer to TH will have a lower error rate $\epsilon_k^{(m)}$ and a larger $\alpha_k^{(m)}$, which results in a larger weight w_i for the samples incorrectly classified as class k (Line 17), thus, helping the precision of class k to pass TH in the next iteration (by correctly classifying such samples with larger weights).

One difference of Algorithm 2 from Algorithm 1 is that our $\epsilon_k^{(m)}$ and $\alpha_k^{(m)}$ are calculated for each class k, since our goal is to push the classes not in $C^{(m)}$ into $C^{(m+1)}$. We use the term *per-class weight* w_i obtained from such $\alpha_k^{(m)}$ to distinguish from Algorithm 1's weights w_i based on the class-blind $\alpha^{(m)}$. Another difference is that $\epsilon_k^{(m)}$ and $\alpha_k^{(m)}$ are calculated on the separate validation set V, as opposed to the training set as in Algorithm 1. This allows us to determine the prediction confidence for class k based on its generalization error instead of training error, to avoid over-fitting.

When the class distribution is imbalanced, the accuracy (in terms of precision and recall) of the minority class is usually more important than that of the majority class, and our per-class weighting can help with this goal. For example, in a training set for predicting the buyer class, 5% of samples have the buyer class and 95% of samples have the non-buyer class. For such datasets, the majority class k often has higher precision $Precision_k$ than the minority class due to the sample dominance, and our per-class weighting will assign a larger weight w_i to the minority class samples x_i misclassified as the majority class, compared to the weight assigned to the majority class samples misclassified as the minority class. This help correctly classifies such samples x_i through the larger weight w_i in the next iteration, thus, increasing the recall of the minority class. We will study the effect of our per-class weighting in addressing this requirement of imbalanced-class distribution in Section 5.3.2.

Remove samples having confident predictions (Line 13-15). The current base classifier $T^{(m)}$ is responsible for classifying the samples that are assigned to a class k from $C^{(m)}$ by $T^{(m)}$. Therefore, such samples are considered "done" and should be removed from the training set S' and validation set V in the next iteration. Note that the remaining samples in S' and V may still have a class k in $C^{(m)}$ as true class; we only remove samples that are *predicted* to have the classes k in $C^{(m)}$. One special case is that $C^{(m)} = \emptyset$, so no sample is removed from S' and V in the next iteration. In this instance, the iterative process will still continue since the weights w_i will be updated at Line 17 because the exponential term is not equal to 1 (note that not all samples are correctly classified, otherwise we do not have $C^{(m)} = \emptyset$).

If $S' = \emptyset$ or $V = \emptyset$ (Line 20-23), the training phase will be terminated and the remaining iterations are ignored by setting $\alpha^{(m+1)} \dots \alpha^{(M)}$ to the zero vector $\mathbf{0}$.

Return the list of base classifiers (Line 26). At the end of training, LinearBoost returns a list L which contains M base classifiers, $(T^{(m)}, C^{(m)}, \alpha^{(m)})$, denoting the base classifier, confident classes, and class weights for iteration m, $1 \leq m \leq M$. The prediction phase (i.e., inference phase) using these classifiers will be covered in Section 3.2.

The followings are two additional options to optimize Algorithm 2.

- *Transfer learning.* Transfer learning [47] is a learning technique that leverages knowledge learned from a related task for a new task. To apply this technique, at iteration m, instead of randomly initializing the base classifier $T^{(m)}$'s parameters at Line 3 of Algorithm 2, we can initialize $T^{(m)}$'s parameters using the learned $T^{(m-1)}$'s parameters and obtain $T^{(m)}$ from fine-tuning $T^{(m-1)}$'s parameters using the current training set. With a better initialization than random initialization, the learning of $T^{(m)}$ can converge faster and require less data.

- *Weight resetting.* As **Compute per-class weights** mentioned, LinearBoost algorithm assigns greater weights w to the samples incorrectly classified into classes with low error rates, to boost those classes' precision to pass `TH`. Once the precision of these classes passes `TH`, LinearBoost's attention should be shifted away from them and give all classes the same chance. Hence, the weights w (Algorithm 2 Line 17) should be reset to uniform weights for the next iteration. This weight resetting serves the purpose of giving all classes the same chance. The weight resetting can be applied every several consecutive iterations. In experiments, we found that resetting weight after every 5 consecutive iterations works well for most datasets.

We will study the effectiveness of these options in Section 5.3.2.

3.2 Prediction Phase

Algorithm 2 returns a list of M base classifiers. Later base classifiers in the list are likely over-fitting because they are trained on smaller training sets due to removing some training samples after each iteration. We can identify an optimal l-length prefix of the list for making prediction, where $l \leq M$, and the optimal prefix that achieves a minimized generalization error will be discussed in Section 4.1. Algorithm 3 presents the prediction phase assuming that a list of l base classifiers, $\{(T^{(1)}, C^{(1)}, \alpha^{(1)}), \ldots, (T^{(l)}, C^{(l)}, \alpha^{(l)})\}$, is used for prediction.

To classify a sample x, this algorithm applies the list of l base classifiers $(T^{(m)}, C^{(m)}, \alpha^{(m)})$ sequentially starting with $m = 1$. If the current base classifier $T^{(m)}$ predicts a class k from $C^{(m)}$, i.e., $T^{(m)}(x) = k \in C^{(m)}$, the prediction $H(x)$ for x is finalized to k. If $T^{(m)}$ predicts a class k not from $C^{(m)}$, i.e., $T^{(m)}(x) = k \notin C^{(m)}$, for all $1 \leq m \leq l$, the prediction $H(x)$ for sample x is determined by the weighted majority voting from all l base classifiers (Line 6). The weighted majority voting uses $\alpha_k^{(m)}$ as the weights, therefore, if the sample x is predicted to a class k at iteration m and the base classifier $T^{(m)}$ has a lower error rate $\epsilon_k^{(m)}$ on the class k, i.e., class k prediction is more confident, the class k prediction will be weighted by a larger $\alpha_k^{(m)}$ in the final prediction.

Algorithm 3 LinearBoost – Prediction phase

```
Input: Sample = $x$; a list of base classifiers $\{(T^{(1)}, C^{(1)}, \alpha^{(1)}), \ldots, (T^{(l)}, C^{(l)}, \alpha^{(l)})\}$.
Output: Predicted class $H(x)$ for $x$.
1: for $m = 1$ to $l$ do
2:     if $T^{(m)}(x) = k \in C^{(m)}$ then
3:         return $H(x) = k$.
4:     end if
5: end for
6: return $H(x) = \underset{k}{\operatorname{argmax}} \sum_{m=1}^{l} \alpha_k^{(m)} \cdot \mathbb{1}(T^{(m)}(x) = k)$.
```

Chapter 4

Several Issues

In Chapter 3, M, the number of base classifiers in L, and `TH`, the threshold for confident prediction, are provided as inputs.This section will address the question of how to determine the number of base classifiers and how to set `TH`.

4.1 Prune the Classifier List

As mentioned in Section 2.1, the bound on the generalization error of AdaBoost is more theoretical than practical. In particular, Theorem 2 assumes certain conditions, and if these conditions are not satisfied, the generalization error is not guaranteed to decrease as the number of base classifiers increases. In other words, using all M base classifiers in the prediction phase may not result in a good generalization error. There is a similar issue with LinearBoost.

To address the above issue, one practical solution is choosing a slightly larger M for the list L of returned base classifiers and identifying a prefix of L to minimize the generalization error on a validation set V. This solution is given in Algorithm 4. Minimizing the generalization error is performed by maximizing the Macro F1. Other performance metrics, such as Weighted F1, can be used instead of Macro F1.

Algorithm 4 Prune the base classifier list

Input: Validation set $V = \{(x_i, y_i)\}$, $i = 1, \dots, n_v$; and $y_i \in \mathbb{C}, \mathbb{C} = \{1, \dots, K\}$; L: the list of base classifiers; M: number of base classifiers in L.

Output: A prefix L_l of base classifiers.

```
1: for $m = 1$ to $M$ do
2:     Run Algorithm 3 with the length-$m$ prefix $L_m$ of $L$ and validation set $V$ to get the
       classification results $H_m$, and calculate the Macro F1 $F1_m$ based on $H_m$.
3: end for
4: return $L_l$ where $l = \underset{m}{\text{argmax}}\ F1_m$.
```

In Algorithm 4, we look for the optimum prefix of L that provides the best Macro F1 on the validation set V. For each length-m prefix L_m of L, where $1 \le m \le M$, we collect the Macro F1, denoted $F1_m$, by applying L_m to V using Algorithm 3. The final ensemble classifier is given by the prefix L_l that has the maximum $F1_l$. The time complexity of Algorithm 4 is $O(M^2 \cdot |V|)$: for each prefix of L, Algorithm 4 runs Algorithm 3 with the validation set V, where each run has the time complexity $O(M \cdot |V|)$.

The $O(M^2 \cdot |V|)$ time complexity of Algorithm 4 can be reduced to $O(M \cdot |V|)$ through Algorithm 5 that computes $H(x)$ for a prefix L_m incrementally from that of the prefix L_{m-1}. Recall from Algorithm 3, either $H(x) = k$ where $k \in C^{(m)}$ is returned by the first base classifier that returns a class in $C^{(m)}$, or $H(x) = \underset{k}{\operatorname{argmax}} \sum_{m=1}^{l} \alpha_k^{(m)} \cdot \mathbb{1}(T^{(m)}(x) = k)$ if none of the base classifiers returns a class in $C^{(m)}$. To compute $H(x)$ for L_m incrementally from that for L_{m-1}, in the first case we set all of $H_m(x), \cdots, H_M(x)$ to k (i.e., Line 5), and in the second case we add $\alpha_k^{(m)} \cdot \mathbb{1}(T^{(m)}(x) = k)$ to the $F_{m-1}(x)$ collected from L_{m-1} (i.e., Line 8), where $F_m(x)$ stores the $H(x)$ for L_m. Note that each base classifier in the input L is applied just once to V. Therefore, the time complexity of Algorithm 5 is $O(M \cdot |V|)$.

Algorithm 5 Prune the base classifier list (less time complexity)

Input: Validation set $V = \{(x_i, y_i)\}$, $i = 1, \ldots, n_v$; and $y_i \in \mathbb{C}, \mathbb{C} = \{1, \ldots, K\}$; L: the list of classifiers $\{(T^{(1)}, C^{(1)}, \alpha^{(1)}), \ldots, (T^{(M)}, C^{(M)}, \alpha^{(M)})\}$; M: number of base classifiers.
Output: A prefix L_l of base classifiers.

```
1: Initialize $F_0(x) = [0, \ldots, 0]$ with $K$ zeros for each $x \in V$.
2: for $m = 1$ to $M$ do
3:   for $x \in V$ do
4:     if $T^{(m)}(x) = k \in C^{(m)}$ then
5:       $H_m(x) = H_{m+1}(x) = \cdots = H_M(x) = k$.
6:       Remove $x$ from $V$.
7:     else
8:       $F_m(x) = F_{m-1}(x) + \alpha_k^{(m)} \cdot \mathbb{1}(T^{(m)}(x) = k)$.
9:       $H_m(x) = \underset{k}{\operatorname{argmax}}\ F_m(x)$.
10:    end if
11:  end for
12:  Calculate Macro F1 $F1_m$ based on $H_m$ and $y$.
13: end for
14: return $L_l$ where $l = \underset{m}{\operatorname{argmax}}\ F1_m$.
```

For a large M, we can apply early stopping with a patience p to Algorithms 4 and 5 in terms of Macro F1 [12]. In our experiments, we use this early stop option with $p = 5$.

4.2 Choice of Precision Threshold `TH`

`TH` is the threshold on the precision for classes with confident predictions (Line 5, Algorithm 2). The selection of `TH` is either manual or automated. In certain mission-critical applications, like medical and security, the user only trusts predictions that pass a certain threshold of confidence. In such a scenario, the user may manually specify a high `TH` value for confident prediction based on domain-specific information or rules and only use those predictions which pass the threshold. For instance, `TH` may be set to 10% more than the known precision for the whole dataset, or to 95%, which is the industry-accepted minimum. In the absence of domain knowledge, `TH` can be searched automatically as a hyperparameter. The search range is $[MP, 1]$, where MP is $T^{(1)}$'s Macro Precision. We pick $T^{(1)}$'s Macro Precision MP as the lowest `TH` value for the search range since $T^{(1)}$ is the initial base classifier without boosting and we expect the confidence of prediction to be at least that of the base classifier without boosting. We consider 5 equally spaced points in the search range $[MP, 1]$ for experiments.

4.3 Comparison

Below, we compare the time complexity of LinearBoost with that of AdaBoost (Table 4.1).

Table 4.1: Time Complexity

Method	Training	Prediction
LinearBoost	$O(M \cdot (\lvert S'\rvert + \lvert V\rvert + \beta) + M \cdot \lvert V\rvert)$	$O(M)$
AdaBoost	$O(M \cdot (\lvert S\rvert + \beta))$	$O(M)$
AdaBoostPruned	$O(M \cdot (\lvert S'\rvert + \beta) + M \cdot \lvert V\rvert)$	$O(M)$

LinearBoost. In Algorithm 2, there are M iterations for training and within each iteration, it needs to train a base classifier $T^{(m)}$ with β time (Line 3) and it needs to go through all training samples $|S'|$ and validation samples $|V|$ to update their weights, therefore, Algorithm 2's time complexity is $O(M \cdot (|S'| + |V| + \beta))$. Applying Algorithm 4 to get an optimal base classifier list, the overall time complexity of LinearBoost is $O(M \cdot (|S'| + |V| + \beta) + M^2 \cdot |V|)$. If we use the more efficient Algorithm 5 to prune the base classifier list, this time complexity can be reduced to $O(M \cdot (|S'| + |V| + \beta) + M \cdot |V|)$.

AdaBoost. Algorithm 1 has the training time complexity $O(M \cdot (|S| + \beta))$. This is same as LinearBoost's training time complexity before applying the pruning algorithm (because $S = S' \cup V$).

AdaBoostPruned. As pointed out in Section 4.1, using all base classifiers returned by AdaBoost in the prediction phase may not guarantee a good generalization error and how to minimize the generalization error is left open. To address this issue, similar to LinearBoost, we can separate S into S' and V and apply Algorithm 4 to determine the best prefix L_l of the list of M base classifiers returned by AdaBoost using the validation set V, and we denote this pruned version of AdaBoost by AdaBoostPruned. Since AdaBoostPruned does not use $C^{(m)}$, we can set $C^{(m)} = \emptyset$ for all base classifiers in Algorithm 4. With using S' as the training set, Algorithm 1 has the training time complexity $O(M \cdot (|S'| + \beta))$ and the training time complexity of AdaBoostPruned is $O(M \cdot (|S'| + \beta) + M^2 \cdot |V|)$, or $O(M \cdot (|S'| + \beta) + M \cdot |V|)$ if Algorithm 5 is used instead of Algorithm 4. Similarly, AdaCost, Asymmetric AdaBoost and AdaC (Section 2.3) can be applied the same procedure to obtain the best prefix of the base classifier list.

Therefore, for the same M and a fixed `TH` (for instance, when `TH` is set manually), the overall time complexity of LinearBoost and AdaBoostPruned is comparable, as the size of the validation set V is typically much smaller than the size of the training set S'. If `TH` is searched using grid search as a hyperparameter, the complexity of LinearBoost will be multiplied by the number of grid search steps for `TH`. On the other hand, in the absence of domain knowledge, AdaBoost's cost sensitive variants, i.e., AdaCost, Asymmetric AdaBoost and AdaC, also require a grid search to identify the optimal cost term. Therefore, the time complexity of the corresponding AdaBoost's cost sensitive variants will also be multiplied by the number of grid search steps.

Finally, the prediction time complexity of LinearBoost, AdaBoost and AdaBoostPruned are on the same level. All of them need to apply a sample sequentially to a list of base classifiers. Each base classifier needs a constant time to get the prediction for the sample, therefore, the prediction time complexity of these methods are relative to the number of base classifiers in the list, i.e., $O(M)$ is the worst case.

Chapter 5

Experiments

We report the empirical evaluation of LinearBoost in this chapter. The experimental setup, including datasets and baselines, is first described. Then, we present the key findings and our analysis.

5.1 Experimental Setup

5.1.1 Datasets

Table 5.1 summarizes the eight datasets used: Fashion MNIST [43], NASA [1], EMNIST Balanced [9], Letter, Synthetic [18], HAR [33], IDA and Shuttle. Letter, IDA and Shuttle come from UCI Machine Learning Repository [11]. These datasets are heterogeneous in terms of real and synthetic datasets, tabular, image and sequence datasets, different data sizes, different feature numbers, different class numbers, balanced-class and imbalanced-class distributions, which provide a comprehensive analysis of the strengths and limitations of various ensemble algorithms in a range of real-world and artificial scenarios. The columns "Train" and "Test" denote the sizes of training and testing samples given in the sources. NASA and HAR do not have the pre-determined testing set so we use the stratified 5-fold cross-validation to split the given dataset into training and testing sets and report the means and standard errors of the 5-fold testing sets [2]. This cross-validation preserves the percentage of samples for each class in each fold. If a validation set V is needed in a method (for example, AdaBoostPruned and LinearBoost), we split the training set S into disjoint S' and V with the ratio 4:1. The column IR represents the *imbalance ratio*, defined as $\frac{C}{C'}$ where C and C' are the maximum and minimum sizes of classes in the number of samples. These eight datasets are grouped into balanced-class datasets that have $IR \leq 1.5$ and imbalanced-class datasets that have $IR > 1.5$. The datasets are detailed in the Appendix.

5.1.2 Baselines

We compare LinearBoost with AdaBoost and its variants. AdaBoost presented in Algorithm 1 is based on SAMME. In our experiments, we use SAMME.R instead of SAMME because

Table 5.1: Dataset Summary

Dataset	Data Type	Class	Feature	Train	Test	IR
Balanced-class Dataset						
Fashion MNITS	Image	10	28x28	60,000	10,000	1.00
NASA	Tabular	2	8	12,220	-	1.00
EMNIST Balanced	Image	47	28x28	112,800	18,800	1.00
Letter	Tabular	26	16	16,000	4,000	1.24
Imbalanced-class Dataset						
Synthetic	Tabular	3	10	12,000	1,333	5.41
HAR	Sequence	18	1800	20,750	-	8.41
IDA	Tabular	2	162	60,000	16,000	59.0
Shuttle	Tabular	5	9	43,483	14,494	883

SAMME.R is the default choice in [30] and performs somewhat better than SAMME and converges faster. Therefore, "AdaBoost" is based on SAMME.R. Also, we additionally compare with AdaBoost's variants which are designed for handling imbalanced-class data, i.e., AdaCost, AdaC2 (AdaC variation), and Asymmetric AdaBoost. For all baselines, we include the pruned versions given by the optimal prefix of the full classifier list as discussed in Section 4.3. We denote them by AdaBoostPruned, AdaCostPruned, AdaC2Pruned, and AsymmetricPruned. We set $M = 20$ iterations for LinearBoost and all baselines. Finally, we include $T^{(1)}$, which represents the single classifier learned by Alg without boosting in our notation $LinearBoost(Alg, Data, M)$, to study the performance gain of multi-iteration boosting methods.

All implementations are coded with Python. AdaBoost, AdaCost, Asymmetric AdaBoost and AdaC2 implementation were transferred from Scikit-learn [30] and imbalanced-ensemble [25] Python library. We use the imbalanced-ensemble Python library's default cost term setting for AdaCost and Asymmetric AdaBoost. The library did not provide the implementation of AdaC series and we implemented AdaC2 by modifying the implementation of AdaCost, and we set each class' cost term to the total number of samples divided by the number of samples in the class. All experiments were conducted on a Windows machine with a processor of 2.8 GHz Intel® Core™ i7-7700HQ CPU (4 Core), 16GB RAM, 1TB SSD storage and an NVIDIA® GeForce® GTX 1060 GPU (6GB).

5.1.3 Training Details for Base Classifiers

The base classifier training is implemented with the backend of PyTorch [29]. Table 5.2 lists the data type, the model structure T of the base classifier and the optimizer's learning rate for each dataset. We use the mini-batches of size 128 for training the base classifier $T^{(m)}$. For each mini-batch, the Adam optimizer is used to update $T^{(m)}$'s parameters [21] with the learning rates from Table 5.2 and the identical weight decay (5e-4) for each dataset. The current training epoch is complete when all the mini-batches have been used for model

Table 5.2: Model Structures and Hyperparameters Used for Each Dataset

Dataset	Data Type	Model T	Learning Rate
Fashion MNIST	Image	CNN	1e-4
NASA	Tabular	MLP	1e-2
EMNIST Balanced	Image	CNN	1e-4
Letter	Tabular	MLP	1e-2
Synthetic	Tabular	MLP	1e-2
HAR	Sequence	RNN	1e-3
IDA	Tabular	MLP	1e-2
Shuttle	Tabular	MLP	1e-2

parameter updating. At iteration m of a boosting method, we train the base classifier $T^{(m)}$ for the maximum 100 epochs using early stopping with a patience p [12]. At epoch e, if the loss on the validation set V (for methods that do not use V like AdaBoost, the loss will be the training set S loss) at the epoch $e - p$ is the lowest among all e epochs ($p = 5$ in our experiment), we use the base classifier learned at epoch $e - p$ as $T^{(m)}$. Intuitively, this base classifier has the lowest loss up to its epoch and this loss cannot be further reduced by the next p consecutive epochs. This early stopping applies to all methods for fitting a base classifier $T^{(m)}$.

The different types of model structures are selected based on different data types of datasets and classification tasks. The details are explained below.

CNN. The CNN model structure is based on the submitted benchmarks in the Fashion MNIST dataset repository[1] and we use the same CNN model structure for the image datasets, i.e., Fashion MNIST and EMNIST Balanced. Before applying the image samples to CNN model, the image pixel values are normalized as suggested in the repository benchmark[2]. The CNN model has the first convolution layer consisting of 32 3x3 filters with 1 padding and 2x2 max-pooling with a stride of 2, the second convolution layer consisting of 64 3x3 filters and 2x2 max-pooling, 1 fully connected hidden layer with 512 neurons, and the output layer with 10 neurons for Fashion MNIST or 47 neurons for EMNIST Balanced. In each convolution layer, we apply Batch Normalization [19], ReLU activation [26] and dropout [35]. ReLU activation and dropout are applied in the fully connected hidden layer. To improve the generalization of the first base classifier $T^{(1)}$, we apply the data augmentation [45] on the first iteration. The specific data augmentation settings for each dataset are as follows: taking a random cropping from images and padding with 4 pixels on 4 sides for both Fashion MNIST and EMNIST Balanced; images in Fashion MNIST are randomly performed horizontal flipping and images in EMNIST Balanced are randomly rotated within

[1]https://github.com/zalandoresearch/fashion-mnist

[2]https://github.com/Queequeg92/DualPathNet/blob/master/train_fashion_mnist.py

25 degrees. The optimizer learning rate for training the first base classifier $T^{(1)}$ is initially set to 5e-4 and dropped to 1e-4 for the remaining iterations.

MLP. Multilayer Perceptron (MLP) is used for the tabular datasets. The MLP model consists of one input layer, two hidden layers of 70 and 50 neurons, and one output layer. Between each layer, the ReLU activation function is used. This MLP model structure was used in [32] as the base classifier of AdaBoost on the Letter dataset. We use the same MLP structure for all tabular datasets. All tabular sample feature values are pre-processed using the standard scaler[3].

RNN. We follow [27] to build a simple Recurrent Neural Network (RNN) for the sequence data HAR. The RNN model has 1 LSTM layer with an input size of 1800 and a hidden layer size of 128. The last layer consists of 18 output neurons. Between the LSTM and output layer, dropout is used. The sequence samples are applied Fast Fourier Transform [10] before applied to the RNN model as [33] suggests.

5.2 Results

5.2.1 Analysis on AdaBoost

Theorem 2 gives the convergence property of AdaBoost that the generalization error decreases as the number of iterations M increases, and we pointed out that this property is of only theoretical value because in practice the required condition on VC-dimension and data size may not hold. In this section, we like to study the impact of M on the performance, i.e., Macro F1 score (defined in Section 2.5), using the eight datasets. To this end, we compare AdaBoost and AdaBoostPruned as boosting strategies, denoted by $\mathit{AdaBoost}(\mathit{Alg}, \mathit{Data}, M)$ and $\mathit{AdaBoostPruned}(\mathit{Alg}, \mathit{Data}, M)$, where Alg is the neural network learning algorithm for training a base classifier defined in Section 5.1.3 and Data is a dataset defined in Section 5.1.1. $\mathit{AdaBoost}(\mathit{Alg}, \mathit{Data}, M)$ returns the full list of M base classifiers, whereas $\mathit{AdaBoostPruned}(\mathit{Alg}, \mathit{Data}, M)$ returns an optimal prefix of the full list using a validation set V reserved from the training data as described in Section 4.3, i.e., AdaBoost uses S as Data to train the ensemble classifier, whereas AdaBoostPruned splits S into S' and V and trains the ensemble classifier using S' as Data.

Figure 5.1 shows the Macro F1 of AdaBoost and AdaBoostPruned on the eight datasets. For every $m \leq M$ (M is the largest value shown on x-axis), the green solid line plots AdaBoost's Macro F1 on the training set and the green dash line plots the Macro F1 on the testing set when the first m base classifiers of AdaBoost are used for prediction. The red dash line displays the testing Macro F1 of AdaBoostPruned. AdaBoostPruned is not necessarily the highest point of AdaBoost because its training set is smaller than that of

[3]https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

AdaBoost due to reserving a validation set V for finding the optimal prefix of the classifier list.

The first finding is that AdaBoost's training performance does not always increase as the number of iterations increases; in other words, the training error does not always decrease. Another finding is that, except for Shuttle, AdaBoostPruned achieves a better performance on the testing set compared with AdaBoost's performance at the final iteration M. This suggests that using more training data in AdaBoost (i.e., S instead of S' as in AdaBoost-Pruned) does not significantly improve the testing Macro F1 in most cases compared with AdaBoostPruned and AdaBoost can not identify which iteration achieves the best testing Macro F1 as it does not have a validation set V to measure the generalization performance. In contrast, after reserving a fraction of training data as a validation set V, AdaBoost-Pruned can identify the best first m base classifiers which achieve the nearly best testing Macro F1 of AdaBoost over $1 \leq m \leq M$. Therefore, using a validation set to determine the optimal iteration number is necessary for AdaBoost.

Based on this study, AdaBoostPruned achieves better testing performance than AdaBoost in most cases. In the rest of our evaluation, we consider only the pruned versions of AdaBoost and its variants, i.e., by AdaBoostPruned, AdaCostPruned, AdaC2Pruned, and AsymmetricPruned.

5.2.2 Main Results

Below, all results are based on the testing set and we employ both transfer learning and weight resetting for LinearBoost (see Section 3.1).

Table 5.3 shows the performance of all methods on the eight datasets. For NASA and HAR, we report the means and standard errors of the 5-fold cross validation. For the other datasets, we report the F1 scores on the given testing set. The best F1 scores are highlighted in bold-face. Across all datasets and the two F1 metrics, LinearBoost outperforms all baselines. The Wilcoxon signed-rank test [42] (one-sided) confirms that these improvements by LinearBoost over AdaBoostPruned on the population of the eight datasets are statistically significant at 5% significance level (with *p-value* $= 0.004$ for both metrics) [41]. Similarly, the improvements of LinearBoost over AdaCostPruned, AdaC2Pruned and AsymmetricPruned are statistically significant (with *p-value* $= 0.004$ for both metrics). The main findings are as follows.

First, LinearBoost performs the best in both Macro F1 and Weighted F1 across the four balanced-class datasets (i.e., Fashion MNIST, NASA, EMNIST Balanced and Letter). LinearBoost's improvements over $T^{(1)}$ are (2.8%, 2.8%) for Macro F1 and Weighted F1 respectively (averaged over the four datasets), whereas such improvements by AdaBoost-Pruned are only (0.21%, 0.21%). As expected, AdaCostPruned, AdaC2Pruned, and AsymmetricPruned have similar improvements as AdaBoostPruned due to the balanced-class distribution.

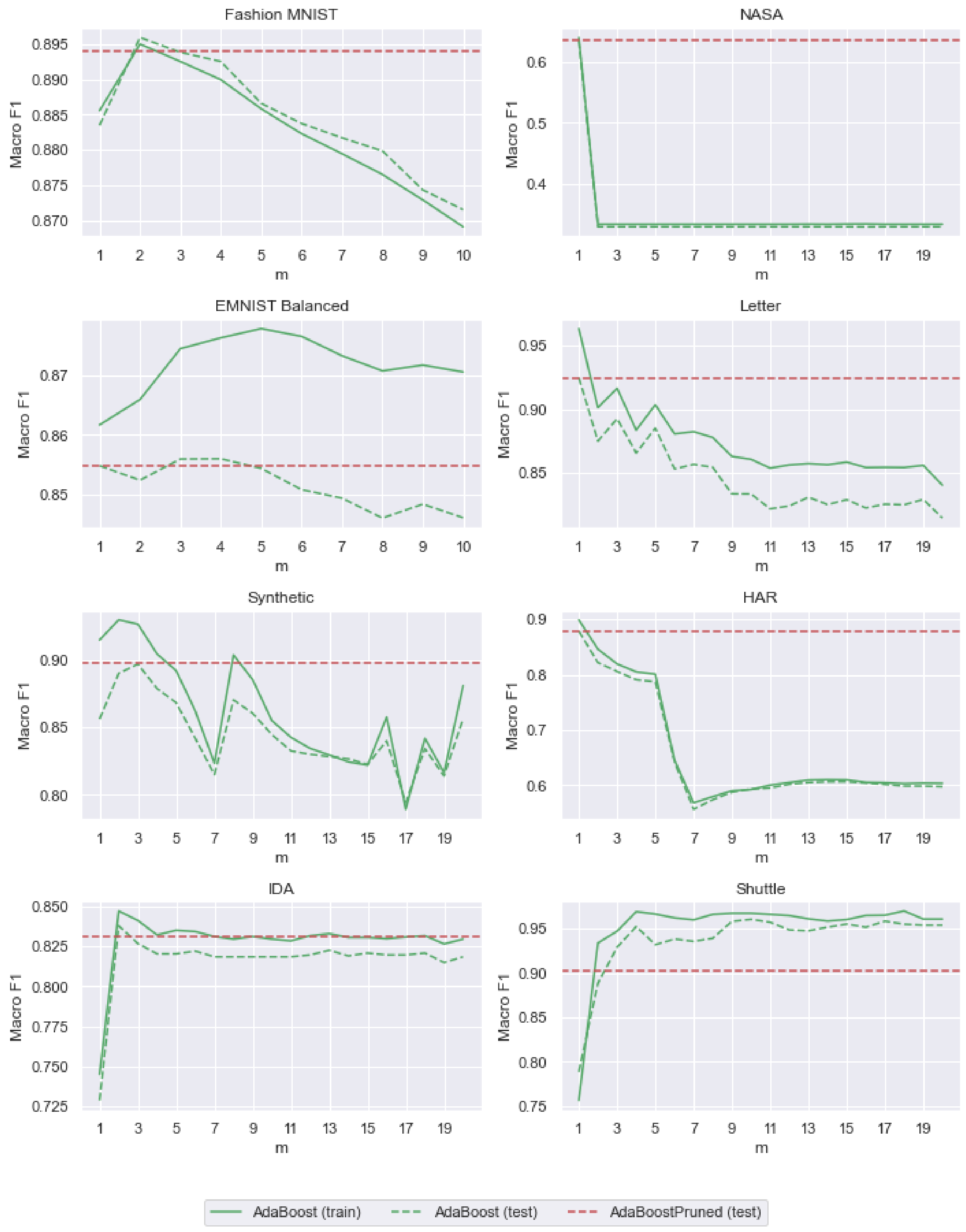

Figure 5.1: Macro F1 of AdaBoost and AdaBoostPruned

Table 5.3: F1 Scores of Eight Datasets

Method	Macro F1	Weighted F1	Macro F1	Weighted F1	Macro F1	Weighted F1	Macro F1	Weighted F1
	Fashion MNIST		NASA		EMNIST Balanced		Letter	
$T^{(1)}$	0.8835	0.8835	0.6368 ± 0.0018	0.6369 ± 0.0019	0.8548	0.8548	0.9249	0.9250
AdaBoostPruned	0.8939	0.8939	0.6348 ± 0.0012	0.6347 ± 0.0012	0.8548	0.8548	0.9249	0.9250
AdaCostPruned	0.8997	0.8997	0.6341 ± 0.0024	0.6341 ± 0.0024	0.8548	0.8548	0.9249	0.9250
AdaC2Pruned	0.9023	0.9023	0.6357 ± 0.0016	0.6356 ± 0.0016	0.8548	0.8548	0.9249	0.9250
AsymmetricPruned	0.8941	0.8941	0.6360 ± 0.0015	0.6360 ± 0.0015	0.8548	0.8548	0.9249	0.9250
LinearBoost	**0.9318**	**0.9318**	**0.6454 ± 0.0047**	**0.6455 ± 0.0046**	**0.8839**	**0.8839**	**0.9498**	**0.9499**
	Synthetic		HAR		IDA		Shuttle	
$T^{(1)}$	0.8560	0.9342	0.8208 ± 0.0292	0.8162 ± 0.0293	0.8311	0.9858	0.7886	0.9976
AdaBoostPruned	0.8972	0.9477	0.8208 ± 0.0292	0.8162 ± 0.0293	0.8311	0.9858	0.9019	0.9983
AdaCostPruned	0.9248	0.9644	0.8991 ± 0.0048	0.9110 ± 0.0058	0.8815	0.9893	0.7542	0.9975
AdaC2Pruned	0.8560	0.9342	0.8208 ± 0.0292	0.8162 ± 0.0293	0.8311	0.9858	0.8831	0.9969
AsymmetricPruned	0.9056	0.9516	0.8208 ± 0.0292	0.8162 ± 0.0293	0.8323	0.9860	0.9414	0.9989
LinearBoost	**0.9322**	**0.9684**	**0.9113 ± 0.0018**	**0.9221 ± 0.0019**	**0.8924**	**0.9906**	**0.9451**	**0.9991**

Second, LinearBoost performs the best in both Macro F1 and Weighted F1 across the four imbalanced-class datasets as well (i.e., Synthetic, HAR, IDA, and Shuttle). LinearBoost's improvements over $T^{(1)}$ are (9.6%, 3.7%) on average for Macro F1 and Weighted F1. In contrast, such improvements over $T^{(1)}$ by the four baselines are: AdaBoostPruned (3.9%, 0.4%), AdaCostPruned (4.1%, 3.2%), AdaC2Pruned (2.4%, 0%) and AsymmetricPruned (5.1%, 0.5%). LinearBoost has a bigger increase in Macro F1 which is due to the fact that eliminated samples in an iteration tend to be from a larger class that can easily exceed the precision threshold `TH`. This can balance the class distribution which allows subsequent training iterations to concentrate on the learning of samples from a smaller class.

Third, in the four imbalanced-class datasets, we see that AdaCostPruned, AdaC2Pruned, and Asymmetric AdaBoostPruned do not always outperform AdaBoostPruned. This highlights the sensitivity of the cost term for these methods, and the choice of the cost term is not always clear.

Figure 5.2 plots the training and testing Macro F1 changing with different M for LinearBoost and the best baseline (i.e., the baseline which achieves the best Macro F1 in Table 5.3 among AdaBoostPruned, AdaCostPruned, AdaC2Pruned and AsymmetricPruned at each dataset) on the eight datasets. As the figure shows, LinearBoost achieves a better training performance and generalization performance compared with the best baseline in all eight datasets. It is worth noting that the training and testing Macro F1 lines of both methods become flat after some M, this is because the Macro F1 score on the validation set V is decreasing after that point, therefore, the pruning algorithm is early stopped (Algorithm 4 and Algorithm 5). In EMNIST Balanced and Letter datasets, the training and testing Macro F1 lines of the best baseline are flat after the first iteration, which shows that the Macro F1 score on the validation set V is decreasing after the first iteration, so the pruning algorithm returns the base classifier list with the first base classifier only as it achieves the best Macro F1 on V.

Table 5.4 displays the testing F1 score for each class of the four imbalanced-class datasets, sorted by the class size in the testing set. We include $T^{(1)}$, LinearBoost, and

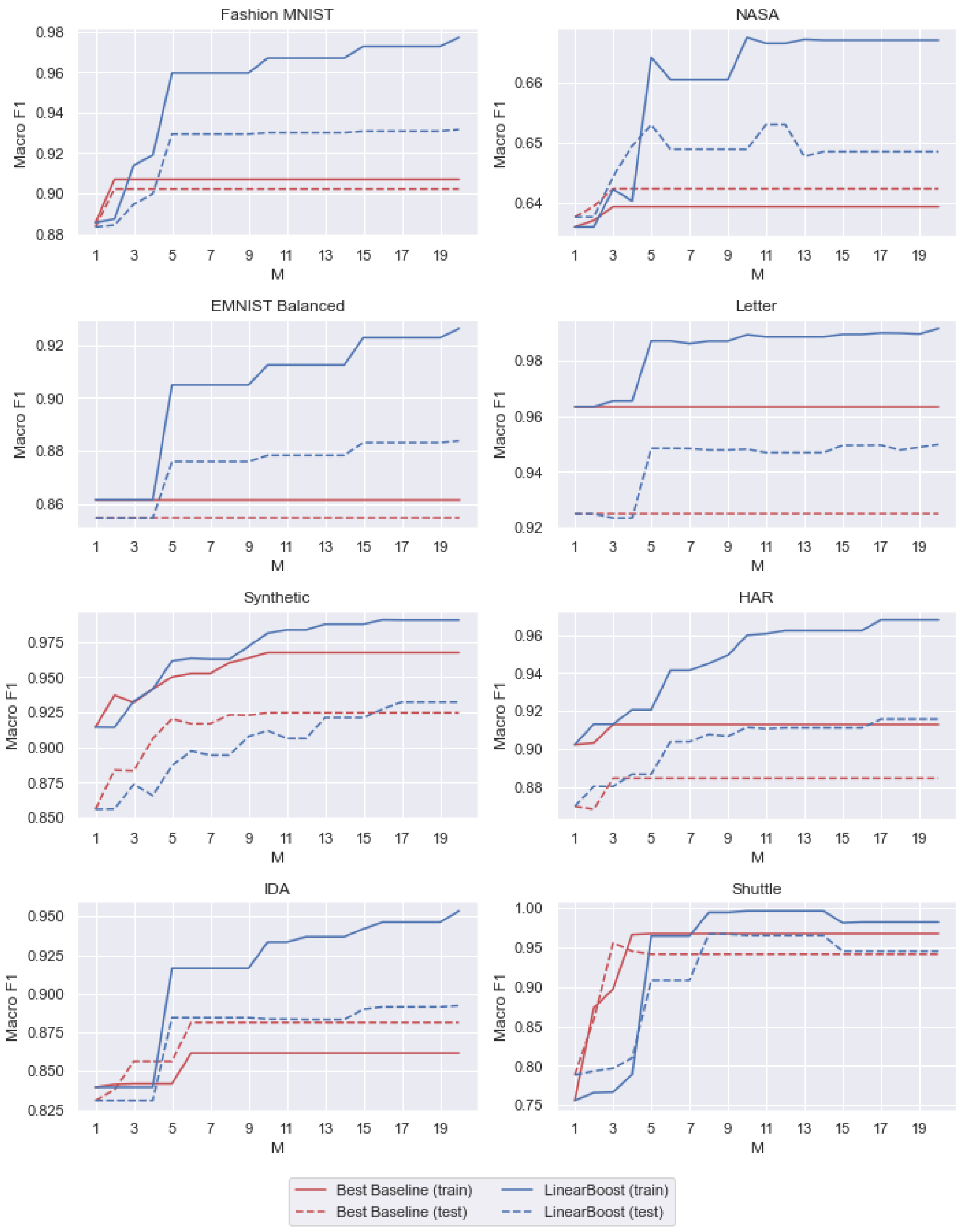

Figure 5.2: Macro F1 of the Best Baseline and LinearBoost

Table 5.4: F1 Scores per Class of the Four Imbalanced-class Datasets

(a) Synthetic

Method\Class	Class 3	Class 1	Class 2
$T^{(1)}$	0.9610	0.9728	0.6344
AdaCostPruned	0.9792	0.9823	0.8127
LinearBoost	**0.9815**	**0.9853**	**0.8299**
Class size	693	512	128

(b) HAR

Method\Class	Stand-sit	Stand	Sit	Talk-stand	Lay	Talk-sit
$T^{(1)}$	0.9443 ± 0.071	0.7311 ± 0.0503	0.6050 ± 0.0951	0.9318 ± 0.0064	0.6035 ± 0.0567	0.8106 ± 0.0299
AdaCostPruned	**0.9632 ± 0.0043**	0.8931 ± 0.0086	0.8507 ± 0.0255	**0.9495 ± 0.0016**	0.8694 ± 0.0354	0.9161 ± 0.0044
LinearBoost	0.9576 ± 0.0076	**0.9161 ± 0.0053**	**0.9014 ± 0.0082**	0.9472 ± 0.0038	**0.9333 ± 0.0078**	**0.9197 ± 0.0018**
Class size	436 ± 4	377 ± 10	375 ± 6	373 ± 5	363 ± 5	359 ± 11
Method\Class	Lay-stand	Pick	Sit-up	Walk	Stair-up	Stair-down
$T^{(1)}$	0.9333 ± 0.0098	0.8670 ± 0.0187	0.8903 ± 0.0188	0.8205 ± 0.0267	0.7522 ± 0.0429	0.7888 ± 0.0457
AdaCostPruned	**0.9562 ± 0.0026**	**0.9079 ± 0.0080**	**0.9350 ± 0.0075**	0.8875 ± 0.0030	0.8565 ± 0.0148	**0.8872 ± 0.0071**
LinearBoost	0.9444 ± 0.0061	0.8967 ± 0.0161	0.9145 ± 0.0181	**0.8883 ± 0.0091**	**0.8863 ± 0.0084**	0.8856 ± 0.0095
Class size	352 ± 3	267 ± 5	201 ± 4	176 ± 5	160 ± 6	156 ± 4
Method\Class	Jump	Run	Push-up	Table-tennis	Walk-backward	Walk-circle
$T^{(1)}$	0.9569 ± 0.0057	0.9498 ± 0.0113	0.9226 ± 0.0159	0.8497 ± 0.0370	0.7095 ± 0.0662	0.7069 ± 0.0476
AdaCostPruned	**0.9768 ± 0.0052**	**0.9672 ± 0.0050**	0.9192 ± 0.0117	**0.9208 ± 0.0105**	0.8200 ± 0.0173	0.7078 ± 0.0311
LinearBoost	0.9624 ± 0.0048	0.9573 ± 0.0099	**0.9393 ± 0.0096**	0.9159 ± 0.0041	**0.8389 ± 0.0103**	**0.7982 ± 0.0156**
Class size	133 ± 2	119 ± 4	96 ± 4	92 ± 4	63 ± 2	52 ± 3

(c) IDA

Method\Class	Negative	Positive
$T^{(1)}$	0.9934	0.6688
AdaCostPruned	0.9946	0.7684
LinearBoost	**0.9955**	**0.7893**
Class size	15,625	375

(d) Shuttle

Method\Class	Rad Flow	Bypass	High	Fpv Open	Fpv Close
$T^{(1)}$	0.9992	0.9986	0.9963	0.8060	0.1429
AsymmetricPruned	**0.9995**	**0.9995**	**0.9994**	0.8286	**0.8800**
LinearBoost	**0.9995**	0.9993	**0.9994**	**0.9091**	0.8182
Class size	11,478	2,155	809	39	13

the best baseline in terms of Macro F1 (Table 5.3) for each dataset. The bold-face denotes the best F1 score for each class.

For Synthetic, LinearBoost achieves the best F1 score across all three classes, with the greatest F1 improvement above $T^{(1)}$ coming from Class 2 which has the smallest class size. AdaCostPruned has the same trend that all classes get improved but the improvements are not larger than LinearBoost's.

For HAR, LinearBoost performs the best in 9 of the 18 classes, and none of LinearBoost's F1 scores is lower than $T^{(1)}$'s. In contrast, AdaCostPruned has one class (Push-up) whose F1 score is slightly lower than $T^{(1)}$'s F1. LinearBoost shows more improvements on the class whose class size is below the average, i.e., minority class. Even for the smallest class, Walk-circle, which only accounts for 1.3% of data, LinearBoost improves the F1 score by 9.1% over $T^{(1)}$, but AdaCostPruned only improves it by 0.1%.

IDA dataset result shows the same pattern as Synthetic dataset. LinearBoost boosts the $T^{(1)}$'s F1 scores for all classes more than AdaCostPruned's and the larger F1 improvement comes from the class with the smallest class size (Positive).

Table 5.5: Precision and the Size of Confident Prediction Produced by LinearBoost

Dataset	Threshold `TH`	Confident %	Precision
Fashion MNIST	0.9720	47.45	0.9810
NASA	1.00 ± 0.0	0 ± 0.0	-
EMNIST Balanced	1.00	3.84	0.9931
Letter	0.9817	33.20	0.9646
Synthetic	1.00	0	-
HAR	0.9174 ± 0.0121	77.89 ± 6.96	0.9409 ± 0.0055
IDA	0.9950	95.33	0.9982
Shuttle	1.00	84.70	0.9999

For Shuttle, $T^{(1)}$, AsymmetricPruned, and LinearBoost have comparable F1 scores for the three biggest classes. In the two smallest classes, AdaBoostPruned has a 2.3% F1 improvement on Fpv Open and 73.7% improvement on Fpv Close's F1, and LinearBoost's improvements are 10.3% on Fpv Open and 67.5% on Fpv Close.

In summary, for imbalanced-class datasets, LinearBoost improves or maintains $T^{(1)}$'s F1 scores on majority classes, whereas minority classes have seen greater improvements. In particular, the average F1 improvements for classes above and below the average class size are 8.5% and 10.9%, respectively. In contrast, the best baseline of each dataset increases the F1 of classes with class sizes above the average by 7.6% (on average), while the F1 improvements of classes with class sizes below the average are 9.8% (on average), which are less than LinearBoost's improvements. This study shows that LinearBoost can emphasize under-performed samples, which are often from minority classes, while maintaining the performance of well-performed samples, which are typically from majority classes.

5.3 Analysis on LinearBoost

In this part, we examine the quality of confident prediction, the various settings and the generality of LinearBoost.

5.3.1 Quality of Confident Prediction

A sample prediction is confident if its predicted class k has a precision of at least `TH` on the validation set (i.e., $k \in C^{(m)}$, see Line 5-7 of Algorithm 2). This `TH` is enforced on the validation set. It remains to see whether a similar performance is enforced on testing samples. Table 5.5 displays the `TH` (determined using grid search on the validation set), the proportion of testing samples with confident predictions (column "Confident %"), and the precision of such samples (column "Precision"). As shown in the table, the precision of testing samples with confident predictions approaches or exceeds the `TH` threshold. The proportion of such testing samples depends on datasets. NASA and Synthetic do not have samples with confident predictions since no class is found to achieve the `TH` $= 100\%$ during

Table 5.6: LinearBoost's F1 Scores of Different Settings

Method	Macro F1	Weighted F1	Macro F1	Weighted F1	Macro F1	Weighted F1	Macro F1	Weighted F1
	Fashion MNIST		NASA		EMNIST Balanced		Letter	
$T^{(1)}$	0.8835	0.8835	0.6368 ± 0.0018	0.6367 ± 0.0019	0.8548	0.8548	0.9249	0.9250
AdaBoostPruned	0.8939	0.8939	0.6348 ± 0.0012	0.6347 ± 0.0012	0.8548	0.8548	0.9249	0.9250
*TR	**0.9334**	**0.9334**	**0.6503 ± 0.0045**	**0.6504 ± 0.0045**	**0.8854**	**0.8854**	**0.9492**	**0.9493**
W*R	0.9217	0.9217	0.6369 ± 0.0018	0.6369 ± 0.0019	0.8720	0.8720	0.9435	0.9435
WT*	0.9079	0.9079	0.6369 ± 0.0023	0.6369 ± 0.0023	0.8548	0.8548	0.9234	0.9235
WTR	**0.9318**	**0.9318**	**0.6454 ± 0.0047**	**0.6455 ± 0.0046**	**0.8839**	**0.8839**	**0.9498**	**0.9499**
	Synthetic		HAR		IDA		Shuttle	
$T^{(1)}$	0.8560	0.9342	0.8208 ± 0.0292	0.8162 ± 0.0293	0.8311	**0.9858**	0.7886	**0.9976**
AdaBoostPruned	0.8972	0.9477	0.8208 ± 0.0292	0.8162 ± 0.0293	0.8311	**0.9858**	0.9019	**0.9983**
*TR	0.9184	0.9625	**0.9100 ± 0.0052**	**0.9193 ± 0.0054**	0.8732	**0.9888**	0.9428	**0.9989**
W*R	0.9047	0.9554	**0.9064 ± 0.0093**	0.9107 ± 0.0106	0.8786	**0.9892**	0.7886	**0.9976**
WT*	0.8925	0.9488	0.8811 ± 0.0072	0.8964 ± 0.0076	0.8741	**0.9889**	**0.9649**	**0.9992**
WTR	**0.9322**	**0.9684**	**0.9113 ± 0.0018**	**0.9221 ± 0.0019**	**0.8924**	**0.9906**	0.9451	**0.9991**

training. Even in this situation, LinearBoost still improves the base classifier $T^{(1)}$'s performance (see Table 5.3). See the discussion in Section 3.1, **Remove samples having confident predictions**.

5.3.2 Ablation Studies

In this section, we study the effectiveness of the three proposed techniques for LinearBoost: per-class weighting scheme (W), transfer learning (T), and weight resetting (R) (proposed in Section 3.1) for LinearBoost. "WTR" denotes employing all three techniques, and * denotes disabling a technique. For example, "*TR" means the setting that transfer learning and weight resetting are enabled but the per-class weighting is disabled; in this case, the original AdaBoost weighting (Algorithm 1 Line 4-6) is used. Table 5.6 shows the results (on the testing set) produced by disabling one technique at a time, i.e., "*TR", "W*R", "WT*", plus the full setting "WTR". The best F1 scores and the second-best F1 scores within 0.5% of the best F1 scores are highlighted in bold-face. The table also includes the performance of $T^{(1)}$ and AdaBoostPruned for comparison.

We see that the full setting "WTR" achieves the best or the second best performance in most cases. "WT*" and "W*R" have the *lowest* average Macro F1 and Weighted F1 across all datasets. This demonstrates that the base classifier training with a good initialization (enabled by transfer learning) and periodically returning to uniform weight for all classes (enabled by weight resetting) are crucial for LinearBoost to generalize well on unseen samples. In details, the average Macro F1 and Weighted F1 of "WT*" on the eight datasets are 0.8670 and 0.8946 respectively, and for "W*R", these averages are 0.8566 and 0.9034. "WTR" achieves 0.8865 Macro F1 and 0.9114 Weighted F1 on average, which are higher than "WT*" and "W*R" settings. The Wilcoxon signed-rank tests (one-sided) confirm that "WTR" Macro F1 and Weighted F1 improvements on the eight datasets are significantly larger than "WT*" and "W*R" improvements at 5% significance level: the *p-value* are 0.020 and 0.008 when comparing "WTR" Macro F1 and Weighted F1 improvements with "WT*"

Table 5.7: IDA $T^{(1)}$'s Validation Set Confusion Matrix

		True	
		Negative	Positive
Predicted	Negative	11,795	82
	Positive	21	102

respectively; the corresponding *p-value* are 0.004 and 0.004 for comparison between "WTR" and "W*R" on Macro F1 and Weighted F1 improvements.

"WTR" Macro F1 and Weighted F1 improvements on all eight datasets are not significantly larger than "*TR" (with *p-value* > 0.05 in both metrics). "WTR" and "*TR" achieves similar performance on the four balanced-class datasets. This suggests that for class-balanced data, using AdaBoost's weighting scheme in LinearBoost is sufficient. However, on the four imbalanced-class datasets, "WTR" achieves 0.9203 Macro F1 on average, which is 1% higher than "*TR" Macro F1 (0.9111 on average). To understand why, let's take the imbalanced-class IDA dataset as an example. Table 5.7 shows IDA $T^{(1)}$'s validation set confusion matrix. Note that AdaBoost's weighting will assign the same $\alpha^{(1)} = 4.75$ to both False Positive and False Negative samples. Since the Negative class (also the majority class) has a larger precision, LinearBoost's per-class weighting will assign a larger weight $\alpha_{neg}^{(1)} = 4.97$ to False Negative samples (i.e., Positive samples misclassified as the Negative class) and assign a smaller $\alpha_{pos}^{(1)} = 1.58$ to False Positive samples. The larger weight assigned to False Negative samples helps correctly classify such samples in the next iteration, thus, increasing the precision of the Negative class. At the same time, correctly classifying these high weight Positive samples leads to an improvement in F1 score of the Positive class.

In summary, all of per-class weighting, transfer learning and weight resetting contribute to improving the LinearBoost's generalization performance on unseen data.

5.3.3 LinearBoost Applied to Other Base Classifiers

LinearBoost is a general methodology that can be applied to *any* base classifier, by calling such learning algorithms as black-boxes, to boost its classification performance. In Section 5.2.2, we have applied LinearBoost to base classifiers defined by different types of neural networks, e.g., CNN, MLP and RNN. To further demonstrate this generality of LinearBoost, we apply LinearBoost to two other different types of base classifiers, i.e., decision trees and XGBoost as base classifiers $T^{(m)}$. XGBoost is the state-of-the-art ensemble method which uses the decision tree as its base classifier (Section 2.4). Table 5.8 shows the hyperparameter settings of base classifiers $T^{(m)}$. The decision tree has the hyperparameter "max_depth" representing the maximum tree depth of the decision tree. XGBoost has two hyperparameters: "max_depth" represents the maximum tree depth of the decision tree used by each XGBoost ensemble classifier, and "gamma" represents a regularization parameter used to control the leaf splitting of such decision tree. The threshold `TH` is determined by grid

Table 5.8: Hyperparameters of Decision Tree and XGBoost as Base Classifiers $T^{(m)}$.

Hyperparameter	Fashion MNIST	NASA	EMNIST Balanced	Letter	Synthetic	HAR	IDA	Shuttle
			Decision Tree $T^{(m)}$					
max_depth	7	10	10	5	13	7	3	1
			XGBoost $T^{(m)}$					
max_depth	2	6	2	6	3	2	3	1
gamma	5	2	5	2	5	15	5	2

Table 5.9: F1 Scores When Decision Tree are Base Classifiers $T^{(m)}$

Method	Macro F1	Weighted F1	Macro F1	Weighted F1	Macro F1	Weighted F1	Macro F1	Weighted F1
	Fashion MNIST		NASA		EMNIST Balanced		Letter	
$T^{(1)}$	0.7592	0.7592	0.7726 ± 0.0041	0.7727 ± 0.0041	0.5307	0.5307	0.3304	0.3319
AdaBoostPruned	0.7592	0.7592	0.7852 ± 0.0034	0.7854 ± 0.0034	0.5307	0.5307	0.5982	0.6008
AdaCostPruned	0.7592	0.7592	0.7840 ± 0.0032	0.7841 ± 0.0032	0.5307	0.5307	0.6396	0.6419
AdaC2Pruned	0.7592	0.7592	0.7862 ± 0.0033	0.7863 ± 0.0033	0.5307	0.5307	0.6085	0.6091
AsymmetricPruned	0.7592	0.7592	0.7858 ± 0.0037	0.7859 ± 0.0037	0.5307	0.5307	0.5573	0.5581
LinearBoost	**0.8234**	**0.8234**	**0.7899 ± 0.0055**	**0.7899 ± 0.0055**	**0.5809**	**0.5809**	**0.7184**	**0.7184**
	Synthetic		HAR		IDA		Shuttle	
$T^{(1)}$	0.5495	0.7175	0.6451 ± 0.0054	0.7193 ± 0.0028	0.7944	0.9832	0.3291	0.8561
AdaBoostPruned	0.6107	0.7775	0.6488 ± 0.0075	0.7015 ± 0.0162	0.8606	0.9879	0.7131	0.9302
AdaCostPruned	0.6236	**0.8147**	0.7002 ± 0.0029	0.7717 ± 0.0029	0.8418	0.9858	0.3291	0.8561
AdaC2Pruned	**0.6267**	0.7576	0.6451 ± 0.0054	0.7193 ± 0.0028	0.8077	0.9795	0.5867	0.7484
AsymmetricPruned	0.6094	0.7711	0.6548 ± 0.0091	0.7050 ± 0.0070	**0.8668**	0.9879	0.8846	0.9312
LinearBoost	0.6094	0.7945	**0.7934 ± 0.0203**	**0.8268 ± 0.0138**	0.8601	**0.9881**	**0.9973**	**0.9999**

Table 5.10: F1 Scores When XGBoost are Base Classifiers $T^{(m)}$

Method	Macro F1	Weighted F1	Macro F1	Weighted F1	Macro F1	Weighted F1	Macro F1	Weighted F1
	Fashion MNIST		NASA		EMNIST Balanced		Letter	
$T^{(1)}$	0.8736	0.8736	0.7742 ± 0.0043	0.7742 ± 0.0043	0.7205	0.7205	0.9052	0.9054
AdaBoostPruned	0.8736	0.8736	0.7825 ± 0.0044	0.7825 ± 0.0044	0.7205	0.7205	0.9052	0.9054
AdaCostPruned	0.8736	0.8736	0.7828 ± 0.0042	0.7828 ± 0.0042	0.7205	0.7205	0.9052	0.9054
AdaC2Pruned	0.8736	0.8736	**0.7830 ± 0.0044**	**0.7831 ± 0.0044**	0.7205	0.7205	0.9052	0.9054
AsymmetricPruned	0.8736	0.8736	0.7813 ± 0.0049	0.7813 ± 0.0049	0.7205	0.7205	0.9052	0.9054
LinearBoost	**0.8822**	**0.8822**	0.7819 ± 0.0029	0.7819 ± 0.0029	**0.7319**	**0.7319**	**0.9225**	**0.9227**
	Synthetic		HAR		IDA		Shuttle	
$T^{(1)}$	0.6320	0.8563	0.8832 ± 0.0008	0.9018 ± 0.0019	0.8756	0.9894	0.9845	0.9997
AdaBoostPruned	0.6795	0.8709	0.8832 ± 0.0008	0.9018 ± 0.0019	0.8814	0.9899	0.9845	0.9997
AdaCostPruned	0.6320	0.8563	0.8832 ± 0.0008	0.9018 ± 0.0019	0.8808	0.9899	0.9845	0.9997
AdaC2Pruned	0.7961	0.9108	0.8870 ± 0.0014	0.9027 ± 0.0017	0.8934	0.9905	0.9845	0.9997
AsymmetricPruned	0.6984	0.8779	0.8832 ± 0.0008	0.9018 ± 0.0019	0.8786	0.9898	0.9845	0.9997
LinearBoost	**0.8185**	**0.9172**	**0.9081 ± 0.0017**	**0.9221 ± 0.0014**	**0.8998**	**0.9909**	**0.9920**	**0.9999**

search. As transfer learning is not commonly used in a tree-based classifier, we disable transfer learning. All methods have the same number of iterations $M = 25$.

Tables 5.9 and 5.10 show the results when base classifiers are decision trees and XGBoost, respectively. The best F1 scores are highlighted in bold-face. As the tables show, LinearBoost improves Macro F1 and Weighted F1 scores of $T^{(1)}$ in all datasets. For decision trees as $T^{(m)}$ (Table 5.9), LinearBoost performs the best in six out of eight datasets, and for XGBoost as $T^{(m)}$ (Table 5.10), LinearBoost performs the best in seven out of eight datasets.

It is true that LinearBoost with decision trees as base classifiers $T^{(m)}$ has worse performance than XGBoost, i.e., Table 5.9 LinearBoost vs Table 5.10 XGBoost $T^{(1)}$'s performance. However, LinearBoost can improve XGBoost by treating the latter as base classifiers, i.e., Table 5.10 LinearBoost vs $T^{(1)}$. This is the advantage of LinearBoost being agnostic to the learning algorithm for base classifiers (by treating such algorithms as a black-box). In

Table 5.11: The *p-value* of the Wilcoxon Signed-rank Tests (one-sided) for Comparing LinearBoost's Macro F1 and Weighted F1 with Those of the Four Baselines

$T^{(m)}$	Metric	AdaBoostPruned	AdaCostPruned	AdaC2Pruned	AsymmetricPruned
Decision Tree	Macro F1	0.020	0.012	0.012	0.020
	Weighted F1	0.004	0.020	0.004	0.004
XGBoost	Macro F1	0.008	0.008	0.008	0.004
	Weighted F1	0.012	0.012	0.020	0.004

contrast, XGBoost cannot apply LinearBoost as its base classifiers because XGBoost is designed specifically for tree base classifiers only.

Table 5.11 shows the *p-value* of the Wilcoxon signed-rank tests (one-sided) for comparing LinearBoost's Macro F1 and Weighted F1 with those of AdaBoostPruned, AdaCostPruned, AdaC2Pruned and AsymmetricPruned, with the population of the eight datasets. At 5% significance level, all *p-value* are smaller than 0.05, suggesting that LinearBoost's improvements on both F1 metrics over these four baselines are statistically significant, for both decision trees as $T^{(m)}$ and XGBoost as $T^{(m)}$. This study suggests that LinearBoost can be applied with different types of base classifiers to improve their performance.

Chapter 6

Conclusions and Future Works

A single classifier may not work well for all classes in a dataset. Class competition and uneven class distribution may explain bad performance in certain classes. In this situation, the ensemble method which learns multiple base classifiers and aggregates their predictions can produce a better result than the single base classifier. The traditional and wildly used ensemble method, AdaBoost, reweights samples during each iteration, giving more weight to samples that were incorrectly classified, and then aggregates all base classifiers to produce a final prediction for each sample. This method pays more attention to samples that were incorrectly classified. However, this kind of reweighting scheme results in poor classifiers for samples that have previously been accurately classified.

Therefore, if a sample is likely to be properly classified, we should use the current prediction rather than the aggregating one as the final result. LinearBoost is the name of the new ensemble algorithm that we propose. The plan is to identify the predictions of samples that have been accurately predicted and *lock in* these predictions in the current iteration, which can be done by using a validation set V and a precision threshold `TH`. If a class k's precision calculated on V is not less than `TH` in the current iteration, LinearBoost considers all samples which are predicted as class k in the current iteration have confident predictions, as LinearBoost has at least `TH` confidence that these predictions are correct. Once the samples are considered as having confident predictions, they will be removed from the training process so that LinearBoost can *concentrate on* other remaining hard-to-classify samples. As the samples with confident predictions are removed during the training, LinearBoost applies *transfer learning* instead of random initializing the base classifier's parameters, which uses the previous fine-tuned base classifier's parameters for initialization. With a good starting point, the base classifier training can converge quickly and require less data to achieve satisfactory performance. In order to maximize the number of classes with confident predictions, LinearBoost computes the *per-class weighting* which can direct the training process to firstly boost the class whose precision is closer to `TH`, i.e., easy to pass `TH`. At the same time, LinearBoost can emphasize the samples from minority classes during the training as LinearBoost's per-class weighting scheme pays more attention to them. However, LinearBoost's per-class weighting scheme can not boost all classes at the same time as it prefers the classes with high precision, therefore, LinearBoost will bring the less-attended classes back to the same level by applying *weight resetting*. For the two key hyperparameters, the number of iterations M and precision threshold `TH`, LinearBoost provides a pruning algorithm to find out M such that achieving the best generalization performance and an automatic way to discover the threshold `TH`.

In the experiment, we compare LinearBoost's performance with AdaBoost and AdaBoost's variants which are designed for imbalanced-class distribution. The results show that LinearBoost has better performance compared with AdaBoost and its variants, and it is able to improve the performance of samples that are poorly-predicted meanwhile maintaining the performance of samples that are well-predicted, according to empirical experiments conducted on datasets with a variety of features. Also, we demonstrate the prediction quality on unseen testing samples with confident predictions is actually high and LinearBoost is a general methodology which can work with any kind of base classifiers, including the state-of-the-art ensemble methods as the base classifiers. The ablation study shows that the combination of *per-class weighting*, *transfer learning* and *weight resetting* can achieve the best performance in most cases. For the time complexity, LinearBoost is comparable with AdaBoost and its variants.

Future Works. The core of the ensemble method proposed in this thesis is to identify the sample predictions with high confidence of being correct in each iteration and "lock in" the predictions to avoid the potential negative effect from later iterations. In future work, we can extend this idea to optimize other ensemble methods. For example, in GradientBoost, the predicted value is updated in the direction of the negative gradient at each iteration. We can set a high threshold for the predicted value and once it reaches the threshold, we have high confidence that the predicted value is correct. We can "lock in" the predicted value and remove the sample in the next procedure. Therefore, the high quality of the sample prediction is maintained.

To get an optimal threshold `TH` automatically, we need to perform a grid search and this is time consuming. Future work includes designing a theoretical way to determine the optimal `TH` that requires less time complexity or improving LinearBoost algorithm so that it does not require `TH` and M as the hyperparameter searching.

We have noticed that some intermediate base classifiers in the base classifier list have negative contributions to the generalization performance, i.e., generalization performance is worse when including these base classifiers (for example, the dropping on the Macro F1, Figure 5.2, NASA and Shuttle datasets). This suggests that carefully selecting the base classifiers returned by LinearBoost's pruning algorithm can further improve the performance. Future work on how to select the most informative base classifiers will be beneficial for not only LinearBoost but also all ensemble methods.

Bibliography

[1] Hamoud Aljamaan and Amal Alazba. Software defect prediction using tree-based ensembles. In *Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering*, PROMISE 2020, page 1–10, New York, NY, USA, 2020. Association for Computing Machinery.

[2] Douglas G Altman and J Martin Bland. Standard deviations and standard errors. *Bmj*, 331(7521):903, 2005.

[3] Peter Bartlett, Yoav Freund, Wee Sun Lee, and Robert E Schapire. Boosting the margin: A new explanation for the effectiveness of voting methods. *The annals of statistics*, 26(5):1651–1686, 1998.

[4] Kevin W. Bowyer, Nitesh V. Chawla, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *CoRR*, abs/1106.1813, 2011.

[5] Leo Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

[6] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[7] Nitesh V Chawla, Aleksandar Lazarevic, Lawrence O Hall, and Kevin W Bowyer. Smoteboost: Improving prediction of the minority class in boosting. In *Knowledge Discovery in Databases: PKDD 2003: 7th European Conference on Principles and Practice of Knowledge Discovery in Databases, Cavtat-Dubrovnik, Croatia, September 22-26, 2003. Proceedings 7*, pages 107–119. Springer, 2003.

[8] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[9] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

[10] James W Cooley and John W Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of computation*, 19(90):297–301, 1965.

[11] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[12] William Falcon and The PyTorch Lightning team. PyTorch Lightning, March 2019.

[13] Wei Fan, Salvatore J Stolfo, Junxin Zhang, and Philip K Chan. Adacost: misclassification cost-sensitive boosting. In *Icml*, volume 99, pages 97–105. Citeseer, 1999.

[14] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[15] Yoav Freund, Robert E Schapire, et al. Experiments with a new boosting algorithm. In *icml*, volume 96, pages 148–156. Citeseer, 1996.

[16] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2):337–407, 2000.

[17] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[18] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

[19] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[20] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30, 2017.

[21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[22] Kewen Li, Guangyue Zhou, Jiannan Zhai, Fulai Li, and Mingwen Shao. Improved pso_adaboost ensemble algorithm for imbalanced data. *Sensors*, 19(6):1476, 2019.

[23] Xuchun Li, Lei Wang, and Eric Sung. Adaboost with svm-based component classifiers. *Engineering Applications of Artificial Intelligence*, 21(5):785–795, 2008.

[24] Weiwei Liu, Ivor W Tsang, and Klaus-Robert Müller. An easy-to-hard learning paradigm for multiple classes and multiple labels. *The Journal of Machine Learning Research*, 18, 2017.

[25] Zhining Liu, Zhepei Wei, Erxin Yu, Qiang Huang, Kai Guo, Boyang Yu, Zhaonian Cai, Hangting Ye, Wei Cao, Jiang Bian, Pengfei Wei, Jing Jiang, and Yi Chang. IMBENS: ensemble class-imbalanced learning in python. *CoRR*, abs/2111.12776, 2021.

[26] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[27] Bolu Oluwalade, Sunil Neela, Judy Wawira, Tobiloba Adejumo, and Saptarshi Purkayastha. Human activity recognition using deep learning models on smartphones and smartwatches sensor data. *arXiv preprint arXiv:2103.03836*, 2021.

[28] David Opitz and Richard Maclin. Popular ensemble methods: An empirical study. *Journal of artificial intelligence research*, 11:169–198, 1999.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Z. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[31] Robert E Schapire. Explaining adaboost. *Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik*, pages 37–52, 2013.

[32] Holger Schwenk and Yoshua Bengio. Boosting neural networks. *Neural computation*, 12(8):1869–1887, 2000.

[33] Niloy Sikder and Abdullah-Al Nahid. Ku-har: An open dataset for heterogeneous human activity recognition. *Pattern Recognition Letters*, 146:46–54, 2021.

[34] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[35] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[36] Yanmin Sun, Mohamed S Kamel, Andrew KC Wong, and Yang Wang. Cost-sensitive boosting for classification of imbalanced data. *Pattern recognition*, 40(12):3358–3378, 2007.

[37] Aboozar Taherkhani, Georgina Cosma, and T Martin McGinnity. Adaboost-cnn: An adaptive boosting algorithm for convolutional neural networks to classify multi-class imbalanced datasets using transfer learning. *Neurocomputing*, 404:351–366, 2020.

[38] Jafar Tanha, Yousef Abdi, Negin Samadi, Nazila Razzaghi, and Mohammad Asadpour. Boosting methods for multi-class imbalanced data classification: an experimental review. *Journal of Big Data*, 7(1):1–47, 2020.

[39] Joaquin Vanschoren. Understanding machine learning performance with experiment databases. *lirias. kuleuven. be, no*, 2010.

[40] Paul Viola and Michael Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. *Advances in neural information processing systems*, 14, 2001.

[41] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M.

Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[42] Frank Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in Statistics: Methodology and Distribution*, pages 196–202. Springer, 1992.

[43] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[44] Qing-Yan Yin, Jiang-She Zhang, Chun-Xia Zhang, and Sheng-Cai Liu. An empirical study on the performance of cost-sensitive boosting algorithms with different levels of class imbalance. *Mathematical Problems in Engineering*, 2013, 2013.

[45] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 13001–13008, 2020.

[46] Ji Zhu, Saharon Rosset, Hui Zou, and Trevor Hastie. Multi-class adaboost. *Ann Arbor*, 1001:48109, 2006.

[47] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.

Appendix A

Dataset Details

Fashion MNIST. Fashion-MNIST is a balanced-class dataset. It comprises 70,000 gray-scale fashion images for classification, measuring 28 by 28 pixels. Each pixel has an integer value between 0 and 255. There are 60,000 training samples and 10,000 testing samples, and each sample belongs to one of ten classes: T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot.

EMNIST Balanced. EMNIST Balanced is a balanced-class dataset and it shares the same image data structure as Fashion MNIST. EMNIST Balanced dataset contains 131,600 images of handwritten digits, lowercase letters and uppercase letters (0 to 9, a to z and A to Z) for classification. There are 112,800 training samples and 18,800 testing samples. The total number of classes is 47 because some letters lowercase and uppercase are merged as one class since they are similar in handwritten, e.g., c/C, i/I, j/J, k/K, l/L, m/M, o/O, p/P, s/S, u/U, v/V, w/W, x/X, y/Y and z/Z.

NASA. NASA dataset is a software defect prediction dataset and it has 11 different versions. In this thesis, we use the preprocessed JM1 version (data cleaning, data balancing with SMOTE [4] and feature selection with Gain Ratio)[1]. This is a balanced-class tabular dataset and each sample has eight numerical features measuring the software statistic. Each sample belongs to Defective or Non-defective class. It has 12,220 samples and doesn't have a pre-determined testing set.

Letter. Letter is a balanced-class tabular dataset with 26 classes (A to Z). This is used for classification tasks and each sample has 16 integer features (statistical moments and counts of edge). It has 16,000 training samples and 4,000 testing samples.

Synthetic. Synthetic dataset has three classes and each sample has ten features which are drawn from a ten-dimensional standard normal distribution. The class of drawn sample x is determined by Equation A.1

[1]https://github.com/hjamaan/PROMISE20-DefectPredictionTreeEnsembles/blob/master/Datasets /Preprocessed/Step3_GainRatio/JM1_Clean_Balanced_GainRatio.csv

$$\text{Class} = \begin{cases} 1, & 0 \leq \sum x_j^2 < \chi^2_{10,\frac{1}{3}} \\ 2, & \chi^2_{10,\frac{1}{3}} \leq \sum x_j^2 < \chi^2_{10,\frac{2}{3}} \\ 3, & \chi^2_{10,\frac{2}{3}} \leq \sum x_j^2 \end{cases} \tag{A.1}$$

where x_j is the j^{th} feature of sample x for $j = 1 \cdots 10$, $\chi^2_{10,\frac{1}{3}}$ and $\chi^2_{10,\frac{2}{3}}$ are the $\left(\frac{1}{3}\right)$ 100% and $\left(\frac{2}{3}\right)$ 100% quantile of the χ^2_{10} distribution, respectively. In this thesis, we generate 12,000 training samples and 1,333 testing samples with a class size ratio of roughly 4:1:5 for Class 1, Class 2 and Class 3. So, Synthetic dataset is an imbalanced-class tabular dataset.

HAR. Human Activity Recognition (HAR) dataset comprises information on 18 distinct classes of activities and includes 20750 samples derived from raw activity samples. This is an imbalanced-class (IR = 8.41), sequence dataset whose each sample contains a total of 1,800 measurements/features on acceleration along the X, Y, Z axes and rate of rotation around the X, Y, Z axes.

IDA. IDA is an imbalanced-class tabular dataset that is used to predict whether failures of trucks related to the Air Pressure system (APS) or not. It has two classes, Positive means the failures are caused by APS and Negative means the failures are not related to APS. Each sample has 170 numerical features and this dataset contains many missing values, so we impute the missing values with median for each feature. For features containing more than 70% missing values, we drop them. We also drop features which contain only a single value throughout all samples. After dropping, each sample has 162 features. The imbalanced ratio of IDA dataset is 59 and this dataset contains 60,000 training and 16,000 testing samples.

Shuttle. Shuttle dataset has seven classes and nine numerical features. The seven classes are Rad Flow, Fpv Close, Fpv Open, High, Bypass, Bpv Close and Bpv Open. We remove Bpv Close and Bpv Open samples in this experiment since these two classes have too few samples for training. The number of training and testing samples of Shuttle dataset are 43,483 and 14,494, respectively. In a word, Shuttle dataset is the most imbalanced-class tabular dataset (IR = 883) in this experiment, roughly 95% of samples belong to Rad Flow or Bypass classes.