Adaptive Boosting Classification

ECEN 5712 Machine Learning Project Report

Mohammed Adib Oumer*

* University of Colorado, Boulder

***Abstract*—I explore adaptive boosting (AdaBoost) algorithm for classification. I chose Adaptive Boosting because it is shown to be robust and relatively immune to overfitting. In my test run, I used Naive Bayes weak learner in a binary classification of pneumonic vs healthy lungs from a set of x-ray images obtained from Kaggle [2]. The images in the dataset were of variable sizes so I scaled all the images to smaller dimensions and applied the classifiers. I tested images of dimension 40x40, 100x100 and 128x128 using a single weak learner and boosted weak learners. Once done with these, I finally worked on multiclass classification of brain tumor MRI classification with data obtained from Kaggle [1]. I also expanded the type of weak learners I used to include logistic regression. The results are nuanced and are discussed in detail later.**

I. Introduction

One of the areas that machine learning (ML) is increasingly being applied to is the medical field. Medical diagnosis could make use of the traditional machine learning application of classification and thereby, enhance detection and treatment opportunities. Particularly, with COVID-19 taking over the world in the last few years, ML can prove to be a very useful and integral part of detecting diseases. Since higher accuracy is desired in medical diagnosis, it is important that we get strong machine learning classifiers and one such simple yet elegant method is boosting. For a test run towards my end goal of multiclass classification, I implement binary classification on x-ray images of normal (healthy) vs pneumonic lungs (obtained from Kaggle) using a Naive Bayes, and decision tree classifier. I chose the pneumonia dataset because it seemed relevant to the current time we are in since pneumonia plays a big role in the case of COVID-19 diagnosis. Once I have the skeleton of the algorithm working, I extend the algorithm to multiclass classification of brain tumor MRI (also obtained from Kaggle). Brain tumor classification plays a niche role in medical prognosis and effective treatment process. Among the fatal diseases, brain cancer is particularly difficult because it is not usually detected until it is too late for prognosis [14]. This project is an attempt to test a realistic relevant implementation of the probabilistic approach to machine learning (covered in the later parts of the course) and gain insight into the algorithms involved. For the custom implementation, this project uses Naive Bayes (Gaussian and multinomial) and logistic regression classifiers as weak learners and applies the popular AdaBoost boosting algorithm. Scikit's implementation of a decision tree is also used as a weak learner within the bone structure of the boosting algorithm for comparison. Deep learning is also an increasingly popular area used in classification so I will compare my results of machine learning algorithm with results from deep learning architectures applied to the same datasets.

II. Theory

A. Naive Bayes

Naive Bayes classifier is a kind of Bayes classifier (classifier based on Bayes theorem). A Bayes classifier is given by [4]:

$$\begin{aligned}\hat{y}(\mathbf{x}) &= arg\max_{y\in\mathcal{Y}} p(y|\mathbf{x}) \propto arg\max_{y\in\mathcal{Y}} p(y)p(\mathbf{x}|y)\\ &= arg\max_{y\in\mathcal{Y}} \mathbf{log}(p(y)) + \mathbf{log}(p(\mathbf{x}|y)) \end{aligned} \quad (1)$$

where the proportionality applies through the Bayes theorem, y is our label and $\mathbf{x} \in \mathbb{R}^d$ is our feature vector test data. Naive Bayes assumes that the features in our feature vector have independent distributions, i.e.:

$$p(\mathbf{x}|y) = \prod_{i=1}^{d} p(x_i|y) \quad (2)$$

For Naive Bayes, I took two approaches on estimating $p(\mathbf{x}|y)$. The first approach is to take the grayscale nature of our image dataset to be a continuous data between [0,1]. This way, each of the probabilities for $p(\mathbf{x}|y)$ can be estimated using normal distributions with the empirical sample mean and sample variances of each class since they are joint maximum likelihood estimates. That is:

$$\begin{aligned} p(y) &= \frac{m_y}{m} \\ p(\mathbf{x}|y) &= \prod_{i=1}^{d} p(x_i|y) \sim \mathcal{N}(\hat{\mu}, \hat{\Sigma}^2) \end{aligned} \tag{3}$$

where m_y is number of class y in the dataset, m is total length of dataset, $\hat{\mu}$ is sample mean vector for each class y and $\hat{\Sigma}^2$ is the sample covariance matrix for each class y, which is a diagonal matrix of the sample variances for each class y [5].

The second approach is to take the feature vectors (pixel values) to be discrete integer values taking values in $\{0, 1, ..., 255\}$ and model them by multinomial distributions. We estimate $p(y)$ as given in Equation 3 but $p(x_i|y)$ is approximated by the smoothed version of the maximum likelihood estimate as given in Equation 4.

$$p(x_i|y) = \frac{N_{yi} + \alpha}{N_y + \alpha m} \tag{4}$$

where N_{yi} is the number of times feature x_i appears in a sample of class y in the training set, $\alpha = 1$ is the Laplace smoothing factor to prevent zero probability calculations, and N_y is the total count of all features for class y [12].

In practical implementations, probabilities can be very small and their products even smaller (potentially causing underflow). As a result, we take the log-likelihood equivalent to the optimization problem shown in Equation 1 using the expressions in Equation 3. Taking the logarithm changes products of probabilities to sums of their logarithms and because log is a monotonically increasing function, the results we get from the log-likelihood is same as the likelihood expression. We can also easily calculate the posterior probability using its log scale version (by taking the exponential of the log).

B. Logistic Regression

Logistic regression is a discriminative learning approach that estimates the posterior probability of the label given the feature vectors in terms of the sigmoid function. The formula is given as:

$$\begin{aligned} \hat{y}(\mathbf{x}) &= arg \max_{y \in \mathcal{Y}} p(y|\mathbf{x}) = arg \max_{y \in \mathcal{Y}} \sigma(y(\mathbf{w}^T\mathbf{x} + b)) \\ &= arg \max_{y \in \mathcal{Y}} \frac{1}{1 + \exp(-y(\mathbf{w}^T\mathbf{x} + b))} \end{aligned} \tag{5}$$

Traditionally, the term logistic regression is used for binary classification (and Equation 5 refers to the binary case), and the multiclass variant is called multinomial logistic regression or softmax regression. Since softmax regression will be able to handle the binary case, from here on my focus will be on softmax regression. One note here is that I will be using the augmented versions $\mathbf{w} \leftarrow [b \ \mathbf{w}^T]^T$ and $\mathbf{x} \leftarrow [1 \ \mathbf{x}^T]^T$ for simplicity of expressions.

The probability in softmax function for multiclass classification (among K classes) is given by the formula [3]:

$$p(y = k|\mathbf{x}) = \frac{\exp(\mathbf{w}_k^T\mathbf{x})}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^T\mathbf{x})} \quad (k = 1...K) \tag{6}$$

The denominator normalizes the numerator terms for each i so that the overall m-sized vector is a PMF. The terms $\{\mathbf{w}_i\}_{i=1}^{K}$ are the parameters needed. Equation 6 basically gives us a probability distribution for each of the classes and we take the class with the maximum probability. To understand why it is called softmax, let's look at this case:

$$(\sigma(\mathbf{z}))_i = \frac{\exp(c \cdot z_i)}{\sum_j \exp(c \cdot z_j)} \tag{7}$$

Note that $c = 1$ gives back the softmax expression. If we take the limit $c \to \infty$, then Equation 7 heads to 1 when $z_i = z_j$, and to 0 otherwise. That is $z_{max} = \max z_i \Rightarrow k = arg \max z_i \Rightarrow (\sigma(\mathbf{z}))_k = 1, (\sigma(\mathbf{z}))_i = 0 \ (i \neq k)$. This is a characteristic of the pure max function. We can see that $c = 1$ is a softened version of the limit that still gives the highest probability for the likely class but does not zero out the least likely outcomes, which is where the term soft applies. The softening plays a critical role in taking the derivatives of the probability while trying to find the parameters $\mathbf{w}$ [9].

So now we have our feature matrix $\mathbf{X} \in \mathbb{R}^{m \times (d+1)}$, our label vector $\mathbf{y} \in \mathbb{R}^m$ and we want to get the parameter matrix $\mathbf{W} \in \mathbb{R}^{(d+1) \times K}$. Then we have:

$$\begin{aligned} p(\mathbf{y}|\mathbf{X}) = l(\mathbf{W}) &= \prod_{i=1}^{m} \frac{\exp(\mathbf{X}_i\mathbf{W}_{y_i=k})}{\sum_{k=1}^{K} \exp(\mathbf{X}_i\mathbf{W}_k)} \\ &\propto \sum_{i=1}^{m} (\mathbf{X}_i\mathbf{W}_{y_i=k} - \log \sum_{k=1}^{K} \exp(\mathbf{X}_i\mathbf{W}_k)) \end{aligned} \tag{8}$$

The proportionality follows through taking the log-likelihood. The log-likelihood is used as the loss function in the gradient ascent algorithm (equivalently the negative log-likelihood in the gradient descent algorithm). The gradient is given as:

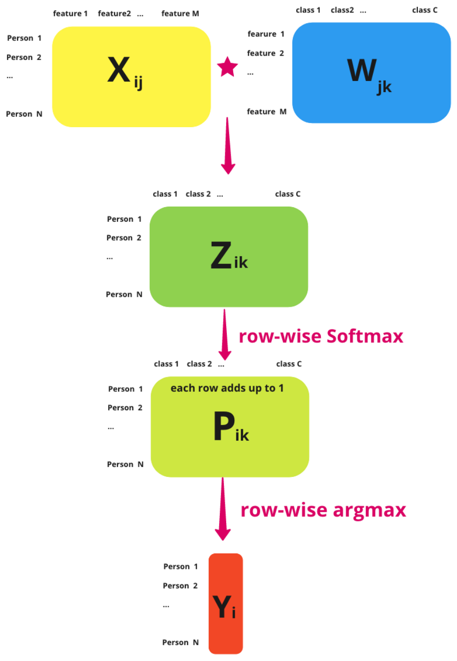

Figure 1. Softmax regression for a matrix feature input

$$\nabla_{\mathbf{W}_k} l(\mathbf{W}) = \sum_{i=1}^{m} \mathbf{X}_i^T \left(\mathbf{I}_{y_i=k} - \frac{\exp(\mathbf{X}_i \mathbf{W}_k)}{\sum_{k=1}^{K} \exp(\mathbf{X}_i \mathbf{W}_k)} \right) \tag{9}$$

We see that the softmax expression shows up in both Equations 8 and 9. We predict the class for a given test vector $\mathbf{x}$ using the $arg\max$ of the probability (shown in Equation 5) calculated using softmax function using the weight parameters and the test vector (shown in Equation 6). Illustrations of the process are shown below in Figures 1 and 2 [16].

C. Boosting

Now that we have our weak classifiers, we need to see how to build a stronger classifier using boosting. The idea behind boosting is that we start by sampling data points, do a classification on them, get the resulting accuracy, increase the sampling weight for misclassified points and iterate over a given number of iterations by sampling data points with their respective updated weights (misclassified points are more likely to be selected) and finally combine results from the several weak learners at every iteration to form a stronger learner. Thus, it is a sequential learning method. An illustration is shown in Figure 3 [15].

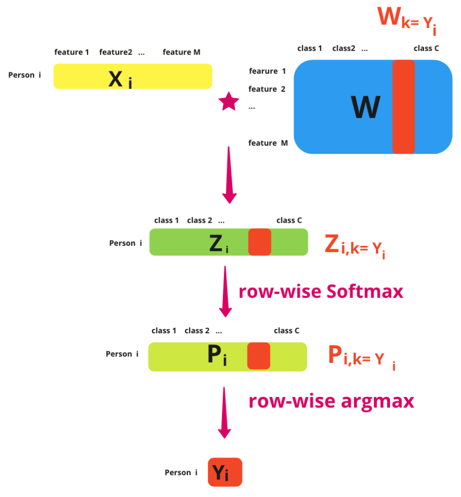

Figure 2. Softmax regression for a vector feature input

The steps in AdaBoost algorithm are given as follows for classifying among K classes [6],[10]:

(a) Initialize the weights $\mathbf{p}_1$ (iteration 1) of all the data points $(\mathbf{x}_i, y_i)$ in our dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ using uniform distribution ($p_i = \frac{1}{m}$).

(b) In each iteration, get m samples from training data with replacement using the weights for each data point as the probability distribution, and find $p(\mathbf{x}|y)$ and $p(y)$ for Naive Bayes and the weights for logistic regression.

(c) Get the predictions using these parameters through Equation 1 and Equation 5. Calculate the total error $e^{(j)}$ for iteration j given as the sum of weights of misclassified points:

$$e^{(j)} = \sum_{i=1}^{m} \mathbb{1}(\hat{y}_i \neq y_i) \times p_i \tag{10}$$

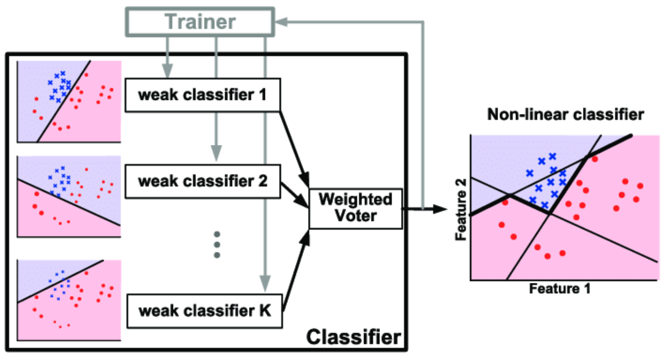

Figure 3. Boosting as combination of weak learners

If the error is worse than a random guess (given by $e^{(j)} > 1 - \frac{1}{K}$), terminate boosting since it may lead to invalid performance measures discussed in the next steps. This termination stop condition seems to be encountered more often as the number of classes increases.

(d) Calculate what is called the performance of the classifier given by $\alpha^{(j)}$:

$$\alpha^{(j)} = \log\left(\frac{1 - e^{(j)}}{e^{(j)}}\right) + \log(K-1) \tag{11}$$

Large $\alpha^{(j)}$ means good classification and small $\alpha^{(j)}$ means poor classification. Thus, using the performance parameter, we can update the weights. $\alpha^{(j)}$ should be positive, which is why we terminate boosting if the error is worse than a random guess in the previous step. If the error is worse than a random guess, $(1 - e^{(j)})$ is smaller than $e^{(j)}$ leading to $\alpha^{(j)}$ being negative. $\alpha^{(j)}$ being negative is detrimental to the next step.

(e) Misclassified points:

$$\mathbf{w}^{(j+1)} = \mathbf{p}^{(j)} \cdot e^{\alpha^{(j)} \times \mathbb{1}(\hat{y}_i \neq y_i)} \tag{12}$$

Properly classified points:
Method 1 [10]:

$$\mathbf{w}^{(j+1)} = \mathbf{p}^{(j)} \tag{13}$$

Method 2 [11]:

$$\mathbf{w}^{(j+1)} = \mathbf{p}^{(j)} \cdot e^{-\alpha^{(j)}} \tag{14}$$

We increase the weights of the wrongly classified records and decrease or keep the weights of the correctly classified records. That is e^{α} makes the updated weight large when the performance α is relatively small (poor classification) and the new weight will be much larger than the old one. And $e^{-\alpha}$ makes the updated weight small when the performance α is relatively large (good classification) and the new weight will be much smaller than the old one (or kept the same, depending on the method for properly classified points). This way, when we choose samples from our training set on the next iteration, we are more likely to choose the misclassified points. To re-iterate, one can notice here that α being negative would have the opposite effect.

(f) A minor implementation detail here is that these updated weights do not necessarily form a probability mass function (pmf) so we normalize each weight by the sum of all the weights at that iteration. That is:

$$\mathbf{p}^{(j+1)} = \frac{1}{\mathbb{1}^T \mathbf{w}^{(j+1)}} \mathbf{w}^{(j+1)} \tag{15}$$

(g) Keep doing this until you reach the maximum iteration M (i.e. $j + 1 = M$). Then finally, we combine the results from all iterations into one final prediction given as:

$$\mathbf{prediction}_{final} = \arg\max_{y} \sum_{j=1}^{M} \alpha^{(j)} \mathbb{1}(\hat{\mathbf{y}}^{(j)} = y) \tag{16}$$

We can then calculate accuracy using our true label and final prediction (for classification: $accuracy = 1 - misclassification\ rate$)

In Equation 16, the exact formulation can change depending on the type of classification. For binary classification, I will use the sign of the above expression (to get class labels in $\mathcal{Y} = \{-1, 1\}$). Also predictions on the validation/test dataset are carried out using the stored performances α, and stored parameters for the classifiers (p_y, the sample means and sample variances for Gaussian Naive Bayes, the probabilities for multinomial Naive Bayes, the weights for logistic regression) and applying Equation 16.

III. RESULTS

The datasets in both the binary and multiclass classification cases were organized in folders with respective labels. The feature engineering aspect here happens in loading the images, resizing each image to custom dimension, saving the resized images, and organizing the training and test data in the form of $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$ where $\mathbf{x}_i \in \{0, 1, ...255\}^d$ is a 1-D reshaped (flattened) version of the image (later rescaled to $[0, 1]$, original dimension $\sqrt{d} \times \sqrt{d}$) and y_i are the labels for each image.

The algorithm was applied for 100 iterations for binary classification (I found this to be reasonable amount of time to run and debug the code). I tried image dimensions of 40x40, 100x100 and 128x128. The results are each shown in Tables I, II and III respectively (accuracy to the nearest integer). The deep learning (DL) results for comparison are shown in Table IV [7].

The images for the multiclass classification were all 512x512 so I resized them down to 64x64 (as much as my computer could take) and carried out the program. I tried iterations of 1, 20, and 50 (equivalent to the maximum number of weak learners applied) and got the respective accuracy shown in Tables V, VI and VII respectively (P.S: the iterations only apply to the

Table I
BINARY: RESULTS FOR 40X40 IMAGES

Iterations = 100	Accuracy (%)		
	Training	Validation	Testing
Single Gaussian Naïve Bayes	85	56	71
AdaBoost Gaussian Naïve Bayes	68	56	72
Single Multinomial Naïve Bayes	86	75	75
AdaBoost Multinomial Naïve Bayes	91	81	79
AdaBoost decision tree (scikit)	99	75	75

Table II
BINARY: RESULTS FOR 100X100 IMAGES

Iterations = 100	Accuracy (%)		
	Training	Validation	Testing
Single Gaussian Naïve Bayes	85	69	72
AdaBoost Gaussian Naïve Bayes	67	69	71
Single Multinomial Naïve Bayes	85	75	74
AdaBoost Multinomial Naïve Bayes	90	81	76
AdaBoost decision tree (scikit)	99	69	73

Table III
BINARY: RESULTS FOR 128X128 IMAGES

Iterations = 100	Accuracy (%)		
	Training	Validation	Testing
Single Gaussian Naïve Bayes	85	69	72
AdaBoost Gaussian Naïve Bayes	67	69	72
Single Multinomial Naïve Bayes	85	75	75
AdaBoost Multinomial Naïve Bayes	87	75	77
AdaBoost decision tree (scikit)	99	81	74

Table IV
BINARY: RESULTS FROM DEEP LEARNING ARCHITECTURES

Architecture: CNN (convolutional neural network)	Accuracy (%)		
	Training	Validation	Testing
Epoch 3	91	56	-
Epoch 6 (learning rate reduced)	95	56	-
Epoch 9 (learning rate reduced)	96	50	-
Epoch 12 (final)	97	88	93

- = cannot be evaluated in the middle of training

Table V
MULTICLASS: RESULTS FOR 64X64 IMAGES

Iterations = 1	Accuracy (%)	
	Training	Testing
Single Gaussian Naïve Bayes	62	58
AdaBoost Gaussian Naïve Bayes	62	59
AdaBoost Multinomial Naïve Bayes	54	49
Single Logistic Regression	66	61
AdaBoost Logistic Regression	70	67
AdaBoost decision tree (scikit)	60	55

AdaBoost rows, not the single pure classifiers). The DL results for comparison are shown in Table VIII.

Table VI
MULTICLASS: RESULTS FOR 64X64 IMAGES

Iterations = 20	Accuracy (%)	
	Training	Testing
AdaBoost Gaussian Naïve Bayes*	71	69
AdaBoost Multinomial Naïve Bayes#	55	56
AdaBoost Logistic Regression	77	72
AdaBoost decision tree (scikit)	72	66

* = 11 elapsed iterations
# = 19 elapsed iterations

Table VII
MULTICLASS: RESULTS FOR 64X64 IMAGES

Iterations = 50	Accuracy (%)	
	Training	Testing
AdaBoost Gaussian Naïve Bayes*	71	69
AdaBoost Multinomial Naïve Bayes#	55	56
AdaBoost Logistic Regression	-	-
AdaBoost decision tree (scikit)	79	73

* = 11 elapsed iterations
# = 19 elapsed iterations
- = Was not able to run it within a reasonable amount of time on my computer

Table VIII
MULTICLASS: RESULTS FROM DEEP LEARNING ARCHITECTURES

Architectures	Accuracy (%)	
	Training	Testing
VGG16 (4 epochs) [8]	95	95
AIFC*	95	95
SIFC*	87	86
ASIFC*	96	95
AIC*	96	93
SIC*	93	91
ASIC*	98	95
CFIC*	99	96

* = from [14]

IV. CONCLUSIONS AND FUTURE WORK

Comparing the pure single learners, the Gaussian model seems to be comparable to the multinomial model in the binary classification case, and the multinomial model seems to be poorer than the Gaussian model in the multiclass classification case. Both Naive Bayes models were very fast (10-20s) so they were a good option to classify. On the other hand, logistic regression which was applied to the multiclass classification dataset was comparable to the Gaussian model but slightly better.

We can see that the boosting algorithm works a bit better on the multinomial and logistic regression than the Gaussian, yet the boost is very minimal. In contrast, on the binary case, the boosting based on Gaussian model diminishes the accuracy on the training set and barely affects the accuracy on the validation and testing set. One explanation could be that the Gaussian model is not a good guess of the distribution of the dataset we have. Another explanation could be that as we take different subsets of the data and fit on them, the predictions vary a lot leading to an averaging of the overall performance and poor prediction. In general, it seems 75% is about as good as we can get with the testing set of the binary dataset. Meanwhile, the dataset for multiclass classification seems to be well built and the characteristics of the predictors is consistent.

Comparing the feature vector size in the binary case, we can see that the results are similar despite having different feature vector sizes. This was the main reason I chose only image of size 64x64 in the multiclass classification case.

The AdaBoost classifier based on scikit's decision trees seems to work well on training data but the validation and testing accuracies are comparable to the AdaBoost classifier based on Gaussian model. The poor performance on the validation and testing sets could be a further indication of the fact that the validation and testing sets are from a different distribution than the training set. Another theoretically expected and empirically proven explanation is that a decision tree is a better weak learner (even weaker the smaller the depth of the tree is) than the given models of Naive Bayes and logistic regression in boosting. Errors in machine learning can be broken down to irreducible(inherent), bias (related to underfitting) and variance (related to overfitting) terms. There is nothing we can do on the irreducible errors. Boosting works better when the base learner is weaker and the base learner is characterized by high variance and low bias. This means every iteration is more likely to lead to different subsets of the data that lead to very different decision tree construction results such that when combined, we get to address all the data points more accurately and get better predictions. In accordance with this, I ran my program for 200 and 500 iterations for multiclass classification with decision trees and was able to get training and testing accuracies of 90% and 86% respectively. On the other hand, both Naive Bayes and logistic regression are a low variance, high bias stable classifiers such that changes in the data points barely affect the results [13]. So it's more likely that we underfit. Hence, the final prediction is more of an averaging of mediocre results over several iterations rather than improving upon previous misclassifications. I am glad to witness these expectations empirically and understand the differences among several learners.

The remarks on the elapsed iterations in Tables VI and VII pertains to the terminating of boosting after the given number of iterations. As mentioned earlier, boosting is terminated when the error is worse than random guess. This seems to occur more often with the multiclass classification case than the binary classification case. So even though the maximum number of iterations specified was 20 and 50, once the weak learners guess worse than random guess, we terminate boosting at the remarked number of iterations (11 for Gaussian Naive Bayes, 19 for multinomial Naive Bayes) and changing the maximum number of iterations does not affect the elapsed number of iterations.

Regarding logistic regression, I used a custom implementation that applies gradient descent algorithm for 100 iterations. This implementation is obviously slow (considering the 100 iterations per logistic regression run and however many logistic regression weak learners I used). This is why Table VII doesn't have values for logistic regression. I compared some of the runs with the logistic regression implementation on scikit that uses smarter and faster optimization algorithms such as stochastic gradient descent and was able to get about 3% accuracy boost for both training and testing accuracies but it still was not as good as the decision tree results.

One other remark is that the way we update the weights of the correctly classified points as shown in Equations 13 and 14 seems to affect the accuracy slightly. The scikit implementation uses Equation 13 and I implemented both options. When I used the same update method as scikit, my results of boosting for all the weak classifiers were similar to the results of the scikit AdaBoost classifier using the same corresponding weak classifiers (except logistic regression, because of the difference in implementation of logistic regression as mentioned in the paragraph above). However, when I used Equation 14 and compared to scikit, I was able to get as much as a 5% boost in the accuracy. I reasoned it could be that the correctly classified points are even more less likely to be selected through Equation 14, thereby resulting in better performances at every weak learner that combine for slightly better results than in the other case.

Lastly, we can see that the DL architectures generally perform much better than boosting in both binary and multiclass classifications. The architectures are quite complex and need good knowledge of setting them up. However, once that is set, the images are fed into them and they output the corresponding results. Unlike boosting (a machine learning algorithm), there is not much tweaking involved in terms of feature engineering and parameter selection. The disadvantage with DL is the complexity of the setup, and the amount of computation resource and time it needs. Yet, it more or less has empirically guaranteed better results. Comparing it to boosting, the boosting algorithm even for decision tree stagnates (similar results for 200 vs 500 weak learners) so it will need more effort and creativity. One observation I made is that while the ML algorithms have theoretical background and require a smart way of thinking to get the best results from the programmer's side, DL mostly delegates that responsibility to the architecture. It was quite a contrast and an intriguing experience.

Further works on this project in terms of boosting could be implementing other boosting algorithms such as Gradient Boosting, LPBoost, and QPBoost. Another classifier option that would likely give good results is K-means neighbor (another high variance, low bias classifier) as a weak learner. One can also explore SVM performance with OvO (one vs one) and OvR (one vs rest) algorithms and explore what works best. Finally, a wild idea would be to combine several weak learners at different iterations and combine several methods such as bagging and boosting, and measure their performances against the DL results.

References

[1] "Brain tumor mri dataset | kaggle," https://www.kaggle.com/datasets/masoudnickparvar/brain-tumor-mri-dataset?resource=download, (Accessed on 05/03/2022).

[2] "Chest x-ray images (pneumonia) | kaggle," https://www.kaggle.com/datasets/paultimothymooney/chest-xray-pneumonia, (Accessed on 05/03/2022).

[3] "Unsupervised feature learning and deep learning tutorial," http://ufldl.stanford.edu/tutorial/supervised/SoftmaxRegression/, (Accessed on 05/04/2022).

[4] C. M. Bishop and N. M. Nasrabadi, *Pattern recognition and machine learning*. Springer, 2006, vol. 4, no. 4.

[5] C. Bustamante, L. Garrido, and R. Soto, "Comparing fuzzy naive bayes and gaussian naive bayes for decision making in robocup 3d," in *Mexican International Conference on Artificial Intelligence*. Springer, 2006, pp. 237–247.

[6] T. Hastie, S. Rosset, J. Zhu, and H. Zou, "Multi-class adaboost," *Statistics and its Interface*, vol. 2, no. 3, pp. 349–360, 2009.

[7] M. Mathur, "Pneumonia detection using cnn(92.6% accuracy)|kaggle," https://www.kaggle.com/code/madz2000/pneumonia-detection-using-cnn-92-6-accuracy, (Accessed on 05/04/2022).

[8] M. Mohaimin, "Brain tumor classification (accuracy ~96%)|kaggle," https://www.kaggle.com/code/mushfirat/brain-tumor-classification-accuracy-96/notebook#6-|-Train-Model, (Accessed on 05/04/2022).

[9] M. A. Nielsen, *Neural networks and Deep Learning*. Determination press San Francisco, CA, USA, 2015, vol. 25.

[10] R. Rojas *et al.*, "Adaboost and the super bowl of classifiers a tutorial introduction to adaptive boosting," *Freie University, Berlin, Tech. Rep*, 2009.

[11] R. E. Schapire, "Explaining adaboost," in *Empirical inference*. Springer, 2013, pp. 37–52.

[12] scikit-learn Developers, "1.9. naive bayes — scikit-learn 1.0.2 documentation." [Online]. Available: https://scikit-learn.org/stable/modules/naive_bayes.html

[13] K. M. Ting and Z. Zheng, "A study of adaboost with naive bayesian classifiers: Weakness and improvement," *Computational Intelligence*, vol. 19, no. 2, pp. 186–200, 2003.

[14] A. Veeramuthu, S. Meenakshi, G. Mathivanan, K. Kotecha, J. Saini, V. Vijayakumar, and V. Subramaniyaswamy, "Mri brain tumor image classification using a combined feature and image-based classifier," *Frontiers in Psychology*, vol. 13, 2022.

[15] Z. Wang, J. Zhang, and N. Verma, "Realizing low-energy classification systems by implementing matrix multiplication directly within an adc," *IEEE transactions on biomedical circuits and systems*, vol. 9, no. 6, pp. 825–837, 2015.

[16] S. Yang, "Multiclass logistic regression from scratch — ph.d. | sr. data scientist," https://sophiamyang.github.io/DS/optimization/multiclass-logistic/multiclass-logistic.html, (Accessed on 05/04/2022).