HW 7

黃熙漢

5. $G(x)$ misclassifies x if at least $M+1$ classifiers misclassify it.

sum all prob at least $M+1$ misclassify

$$E_{out}(G) = \sum_{k=M+1}^{2M+1} \sum_{S \subseteq \{1,2,\dots,2M+1\},\ |S|=k} \left(\prod_{t\in S} e_t\right)\left(\prod_{t\notin S}(1-e_t)\right)$$

Assumed that the classifier are ordered in descending order of it error rates.

$$e_1 \geq e_2 \geq \dots \geq e_{2M+1}$$

upper bound expressed by:

$$E_{out}(G) \leq \sum_{k=M+1}^{2M+1} \binom{2M+1}{k}\left(\prod_{t=1}^{k} e_t\right)\left(\prod_{t=k+1}^{2M+1}(1-e_t)\right)$$

* for minimize $E_{out}(G)$, ensure that individual classifiers have low e_t,
  build on this, Increasing M can reduce $E_{out}(G)$

* independence assumption quite importane

6. update weight

$$u(t+1)_n = u(t)_n \cdot e^{\alpha_t y_n g_t(x_n)}$$

correct classified $\exp(y_n g_t(x_n)=1)$

$$u_n^{(t+1)} = u_n^{(t)} \exp(-\alpha_t)$$

misclassify examples:

$$u_n^{(t+1)} = u_n^{(t)} \exp(\alpha_t)$$

$$U_{t+1} = \sum_{n=1}^{N} u_n^{(t+1)} = \sum_{n: g_t(x_n)=y_n} u_n^{(t+1)} + \sum_{n: g_t(x_n)\neq y_n} u_n^{(t+1)}$$

$$\frac{U_{t+1}}{U_t} = (1-\epsilon_t)\exp(-\alpha_t) + \epsilon_t \exp(\alpha_t)$$

known: $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

$$\exp(\alpha_t) = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}, \quad \exp(-\alpha_t) = \sqrt{\frac{\epsilon_t}{1-\epsilon_t}}$$

$$\frac{U_{t+1}}{U_t} = (1-\epsilon_t)\sqrt{\frac{\epsilon_t}{1-\epsilon_t}} + \epsilon_t\sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$$

$$(1-\epsilon_t)\sqrt{\frac{\epsilon_t}{1-\epsilon_t}} = \sqrt{\epsilon_t(1-\epsilon_t)}$$

$$\epsilon_t\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = \frac{\epsilon_t\sqrt{1-\epsilon_t}}{\sqrt{\epsilon_t}} = \sqrt{\epsilon_t(1-\epsilon_t)}$$

$$\frac{U_{t+1}}{U_t} = 2\sqrt{\epsilon_t(1-\epsilon_t)}$$

$\therefore$ since $0 < \epsilon_t < \frac{1}{2}$, $\epsilon_t(1-\epsilon_t) < \frac{1}{4}$,

$$\frac{U_{t+1}}{U_t} < 1$$

U_t decrease at every iteration at a rate dictated by the weighted error ϵ_t, this underlies the convergence of AdaBoost within $O(\log N)$ interation.

7. $L(y, F(x)) = \frac{1}{2}(y - F(x))^2$

Negative Gradiene $= y_i - F_0(x_i) = y_i$

Fit the base learner

$h_1(x) = X(X^TX)^{-1}X^Ty = Py$, base learer is linear reg

X (design matrix), $P = X(X^TX)^{-1}X^T$

Determine α:

$$F_1(x) = F_0(x) + \alpha_1 h_1(x) = \alpha_1 h_1(x)$$

$$\min . \ L = \frac{1}{2}\sum_{i=1}^{n}(y_i - \alpha_1 h_1(x_i))^2$$

$$\frac{\partial L}{\partial \alpha_1} = -\sum_{i=1}^{n} (y_i - \alpha_1 h_1(x_i)) h_1(x_i) = 0$$

$$\alpha_1 = \frac{\sum_{i=1}^{n} y_i h_1(x_i)}{\sum_{i=1}^{n} h_1(x_i)^2}$$

$h_1(x) = Py$!

$$\alpha_1 = \frac{y^T P y}{y^T P^T P y} = \frac{y^T P y}{y^T P y} = 1$$

$\therefore$ linear reg perfectly fit the negative gradient in first iteration, therefore, scaling $h_1(x)$ by $\alpha_1 = 1$ aligns the model perfectly with the data, minimizing the loss ~~eff~~ effectively.

8.

$$s_n^{(t)} = \sum_{i=1}^{t-1} \alpha_i g_i(x_n)$$

$$L(\alpha) = \sum_{n=1}^{N} (y_n - (s_n^{(t)} + \alpha g_t(x_n)))^2$$

We need
$$\alpha_t = \arg\min_\alpha L(\alpha)$$

$$\frac{\partial L}{\partial \alpha} = \sum^{N} 2(y_n - (s_n^{(t)} + \alpha g_t(x_n)))(-g_t(x_n)) = 0$$

$$\sum_{n=1}^{N} (y_n - s_n^{(t)} - \alpha g_t(x_n)) g_t(x_n) = 0$$

thu

$$\alpha = \frac{\sum_{n=1}^{N} (y_n - s_n^{(t)}) g_t(x_n)}{\sum_{n=1}^{N} g_t(x_n)^2}$$

$$s_n^{(t+1)} = s_n^{(t)} + \alpha_t g_t(x_n)$$

$$y_n - s_n^{(t+1)} = y_n - s_n^{(t)} - \alpha_t g_t(x_n)$$

consider $\sum_{n=1}^{N} (y_n - s_n^{(t+1)}) g_t(x_n)$

$$\sum (y_n - s_n^{(t)} - \alpha_t g_t(x_n)) g_t(x_n)$$

$-$

$$\sum_{n=1}^{N} (y_n - s_n^{(t)}) g_t(x_n) = \alpha_t \sum_{n=1}^{N} g_t(x_n)^2$$

$$\sum_{n=1}^{N} (y_n - s_n^{(t+1)}) g_t(x_n) = \alpha_t \sum_{n=1}^{N} g_t(x_n)^2$$

$$\sum_{n=1}^{N} (y_n - s_n^{(t+1)}) g_t(x_n) = \sum^{N} (y_n - s_n^{(t)}) g_t(x_n)$$

$$- \alpha_t \sum_{n=1}^{N} g_t(x_n)^2$$

substitute $\sum (y_n - s_n^{(t)}) g_t(x_n) = \sum \alpha g_t(x_n)^2$

$$\sum_{n=1}^{N} (y_n - s_n^{(t+1)}) g_t(x_n) = \alpha_t \sum_{n=1}^{N} g_t(x_n)^2 - \alpha_t \sum^{N} g_t(x_n)^2$$
$$= 0$$

$\therefore$ This shown that after updating the model s_n using the optimally chosenstep size α, the new residual $(y_n - s_n)$ become orthogonal to the direction defined by the new learner $(g_t(x_n))$.

Geometric interpretation ; at each step of gradient boost, the pdated residual vector becomes orthogonal to the direction you just moved in.

9.

We want show that any training iteration:

$$w^{(1)}_{i1} = w^{(1)}_{i2} = \dots = w^{(1)}_{id}$$

$$z_i(0) = 0.5 \sum_{j=1}^{d^0} x_j$$

gradient update for weight $w^{(1)}_{i,j}$

$$\Delta w^{(1)}_{i,j} = -\eta \frac{\partial L}{\partial w^{(1)}_{i,j}}$$

$$\frac{\partial L}{\partial w^{(1)}_{i,j}} = \sum \frac{\partial L}{\partial a^{(1)}_i(p)} \frac{\partial a^{(1)}_i(p)}{\partial w^{(1)}_{i,j}}$$, for any training pattern (p)

$$a^{(1)}_i(p) = \tanh\left(\sum_{j=1}^{d^{(0)}} w^{(1)}_{i,j} x^{(p)}_j\right)$$

$$\frac{\partial a^{(1)}_i(p)}{\partial w^{(1)}_{i,j}} = (1 - a^{(1)}_i(p)^2)\, x^{(p)}_j ,$$ cause $\frac{d}{dz}\tanh(z) = 1 - \tanh(z)^2$

$$\frac{\partial L}{\partial w^{(1)}_{i,j}} = \sum_p \frac{\partial L}{\partial a^{(1)}_i(p)} (1 - a^{(1)}_i(p)^2)\, x^{(p)}_j$$

Let $\delta^{(1)}_i(p) = \frac{\partial L}{\partial a^{(1)}_i(p)} \cdot (1 - (a_i(p))^2)$

$$\frac{\partial L}{\partial w^{(1)}_{i,j}} = \sum_{p \in \text{batch}} \delta^{(1)}_i(p) \cdot x_j(p)$$

$$\delta_i^{(1)}(p) = \sum^{d} \frac{\partial L}{\partial z_h^{(2)}(p)} \cdot w_{h,i}^{(2)}(t) \cdot (1 - \alpha_i^{(1)}(p))^2)$$

since $\frac{\partial L}{\partial z_h^{(2)}(p)}$ $w_{h,i}^{(2)}(t)$ are identical across i ,

$$\frac{\partial L}{\partial w_{i,j}^{(1)}} = \frac{\partial L}{\partial w_{i,j+1}^{(1)}}$$

$$\frac{\partial L}{\partial w_{h,i}}(t) = \frac{\partial L}{\partial w_{h,j}^{(1)}}(t) \quad \forall i, j$$

$$w_{h,i}^{(2)}(t+1) = w_{h,j}^{(2)}(t+1) \quad \forall i, j, h$$

$\therefore$ this proof underscores the critical importance of symmetry breaking in neural network training. If all weights are initialized identically and the network structure is symmetric, the network fail to learn diverse and meaningful representation because identical neurons receive identical updates and thus remain identical.

13. LTU

$h(x) = \text{sign}(w \cdot x + b)$

XOR func

$XOR(x_1, x_2 \ldots x_d) = x_1 \oplus x_2 \oplus \ldots \oplus x_d$

1, if an odd number of inputs are 1.

0, otherwise

Prove that no $d-d-1-1$ feedforward neural network with LTUs can compute the Parity function for any $d \geq 0$.

$\because$ $d-1$ hidden neuron, the number of possible activations patterns is at most:

2^{d-1}

output Layer

$f(h) = \text{sign}(v \cdot h + c)$

Parity function requrie:

exactly half of the inputs patterns map to 1 (odd num of 1s)

other map to 0 (even 1s)

For d input bit, there are 2^d possible input combinations. The Parity function assigns $^{d-1}$ of these to 1 and 2^{d-1} to 0.

However, here the crux, each hidden LTU hyperplane divides the input space, but the Parity function requires a highly interdependent mapping that changes output with every single bit flip and with only d-1 hidden LTU, it's impossible to create a unique activation pattern for every single bit flip required by the parity function.

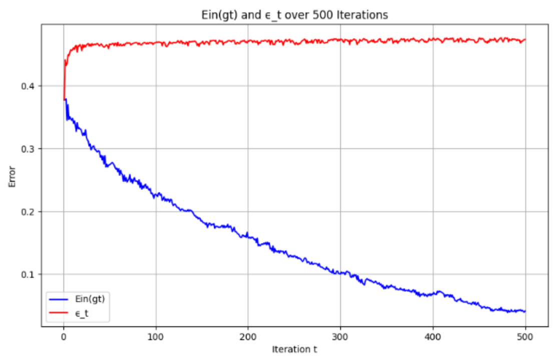
10.

Desribe:

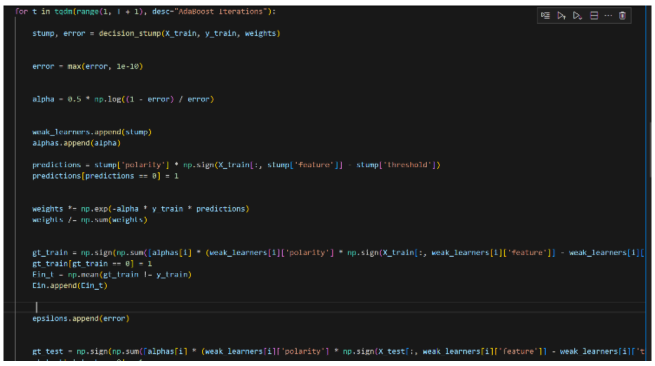
After running the AdaBoost-Stump algorithm for 500 iterations on the Madelon dataset, observe that the in-sample error Ein(Gt) consistently decreases over time. This trend indicates that the ensemble classifier Gt(x) becomes increasingly accurate on the training data as more decision stumps are added. Each iteration of AdaBoost focuses on the misclassified examples from the previous round, allowing the ensemble to correct its mistakes and improve its overall performance.

Concurrently, the weighted error ϵt of the individual decision stumps initially increases and then stabilizes around a certain value. This behavior can be attributed to the adaptive nature of AdaBoost, which reallocates higher weights to the harder-to-classify examples in each subsequent iteration. As the ensemble becomes better at classifying the easier instances, the remaining misclassified examples become more challenging, leading to higher weighted errors for the new stumps focused on these difficult cases. The stabilization of ϵt suggests that the algorithm has reached a point where the remaining errors are consistently difficult to correct, and further iterations no longer significantly alter the weighted error.

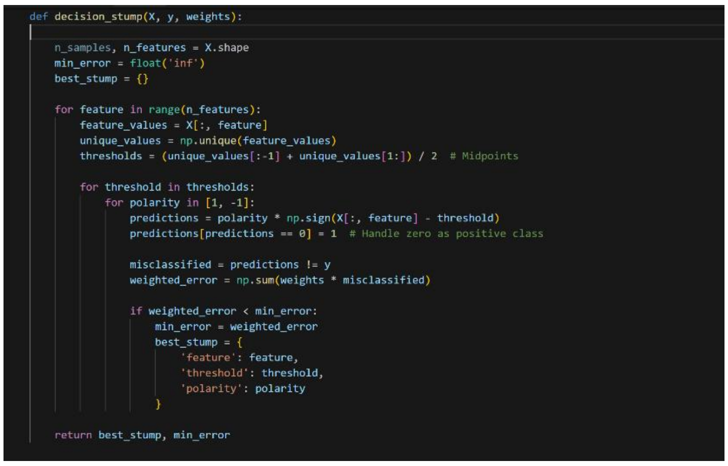
Result:



Code:



```
for t in tqdm(range(1, T + 1), desc="AdaBoost Iterations"):

    stump, error = decision_stump(X_train, y_train, weights)


    error = max(error, 1e-10)


    alpha = 0.5 * np.log((1 - error) / error)


    weak_learners.append(stump)
    alphas.append(alpha)

    predictions = stump['polarity'] * np.sign(X_train[:, stump['feature']] - stump['threshold'])
    predictions[predictions == 0] = 1


    weights *= np.exp(-alpha * y_train * predictions)
    weights /= np.sum(weights)


    gt_train = np.sign(np.sum([alphas[i] * (weak_learners[i]['polarity'] * np.sign(X_train[:, weak_learners[i]['feature']] - weak_learners[i][
    gt_train[gt_train == 0] = 1
    Ein_t = np.mean(gt_train != y_train)
    Ein.append(Ein_t)


    epsilons.append(error)


    gt_test = np.sign(np.sum([alphas[i] * (weak_learners[i]['polarity'] * np.sign(X_test[:, weak_learners[i]['feature']] - weak_learners[i]['t
```



```
def decision_stump(X, y, weights):

    n_samples, n_features = X.shape
    min_error = float('inf')
    best_stump = {}

    for feature in range(n_features):
        feature_values = X[:, feature]
        unique_values = np.unique(feature_values)
        thresholds = (unique_values[:-1] + unique_values[1:]) / 2  # Midpoints

        for threshold in thresholds:
            for polarity in [1, -1]:
                predictions = polarity * np.sign(X[:, feature] - threshold)
                predictions[predictions == 0] = 1  # Handle zero as positive class

                misclassified = predictions != y
                weighted_error = np.sum(weights * misclassified)

                if weighted_error < min_error:
                    min_error = weighted_error
                    best_stump = {
                        'feature': feature,
                        'threshold': threshold,
                        'polarity': polarity
                    }

    return best_stump, min_error
```

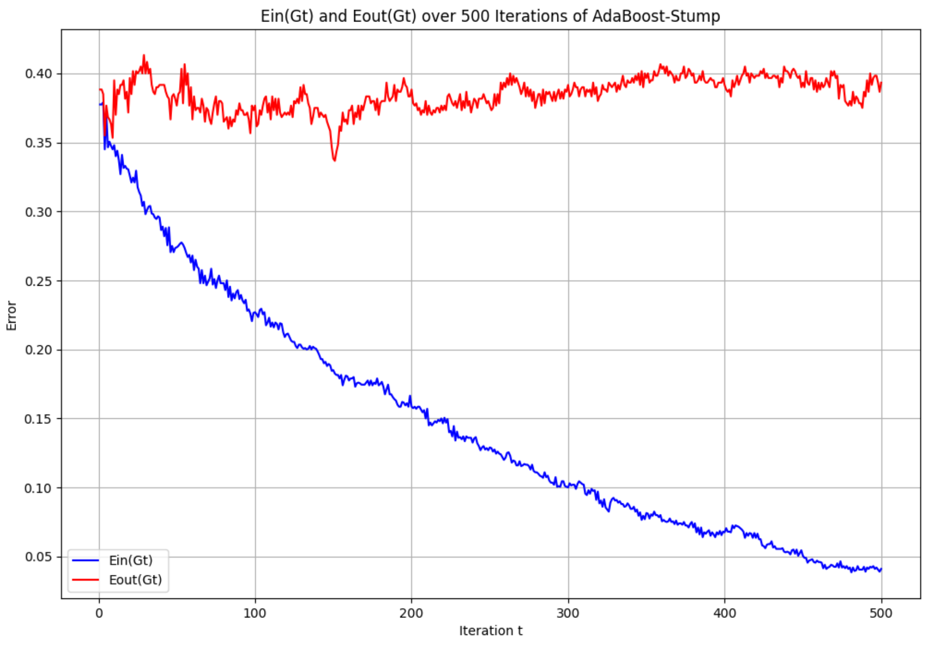
11.

Descibe: The observed behavior, where the in-sample error (Ein(G_t)) consistently decreases while the out-of-sample error (Eout(G_t)) doesn't show a clear initial decline and instead fluctuates within a certain range, deviates from what AdaBoost theory predicts. According to the theory, AdaBoost should not only reduce the training error by focusing on the hard-to-classify instances but also enhance generalization, thereby decreasing Eout(G_t) as the number of iterations increases. I have two perspectives on why this might be happening:

First, AdaBoost is highly sensitive to noisy data and outliers because it continuously adjusts the weights to concentrate on the most challenging instances to classify. If the dataset contains mislabeled examples or inherent noise, the algorithm may overfit these problematic instances. This overfitting drives the in-sample error down but fails to improve the out-of-sample error. As a result, the out-of-sample error may plateau or oscillate because the model becomes too tailored to the specific quirks of the training data, rather than capturing the underlying general patterns.

Second, the Madelon dataset is a synthetic and challenging dataset designed for feature selection and classification. It might have characteristics that aren't fully compatible with the AdaBoost-Stump approach. For example, if the true decision boundary is highly non-linear or requires interactions between multiple features, decision stumps may be insufficient to capture the necessary complexity. This limitation can lead to only modest improvements in Eout(G_t). By considering these factors, it becomes clearer why the out-of-sample error isn't behaving as theoretically expected when using AdaBoost on the Madelon dataset.

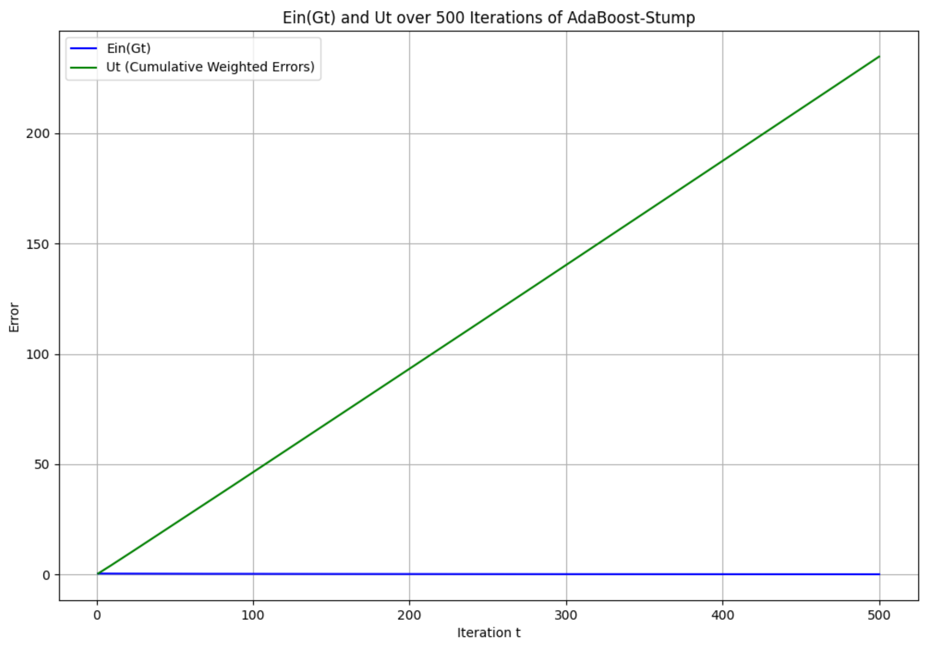
Result:


Ein(Gt) and Eout(Gt) over 500 Iterations of AdaBoost-Stump
Error
Iteration t
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0
100
200
300
400
500
Ein(Gt)
Eout(Gt)

12.

Descibe: after iteration , Ut has a obvious increase without a upper bound , this result shown that At the start of the AdaBoost procedure, weights are initialized evenly across all samples. As successive weak learners are trained, AdaBoost increases the relative weights of the samples that are misclassified to encourage future learners to focus on these harder cases. In principle, this reweighting can drive some sample weights quite high if they remain difficult to classify. Since each round of boosting is normalizing the distribution but also potentially amplifying the weights on a subset of the hardest examples, the *cumulative sum* of updates may appear to grow significantly without an apparent upper bound .

Result :


Ein(Gt) and Ut over 500 Iterations of AdaBoost-Stump
Ein(Gt)
Ut (Cumulative Weighted Errors)
Error
Iteration t