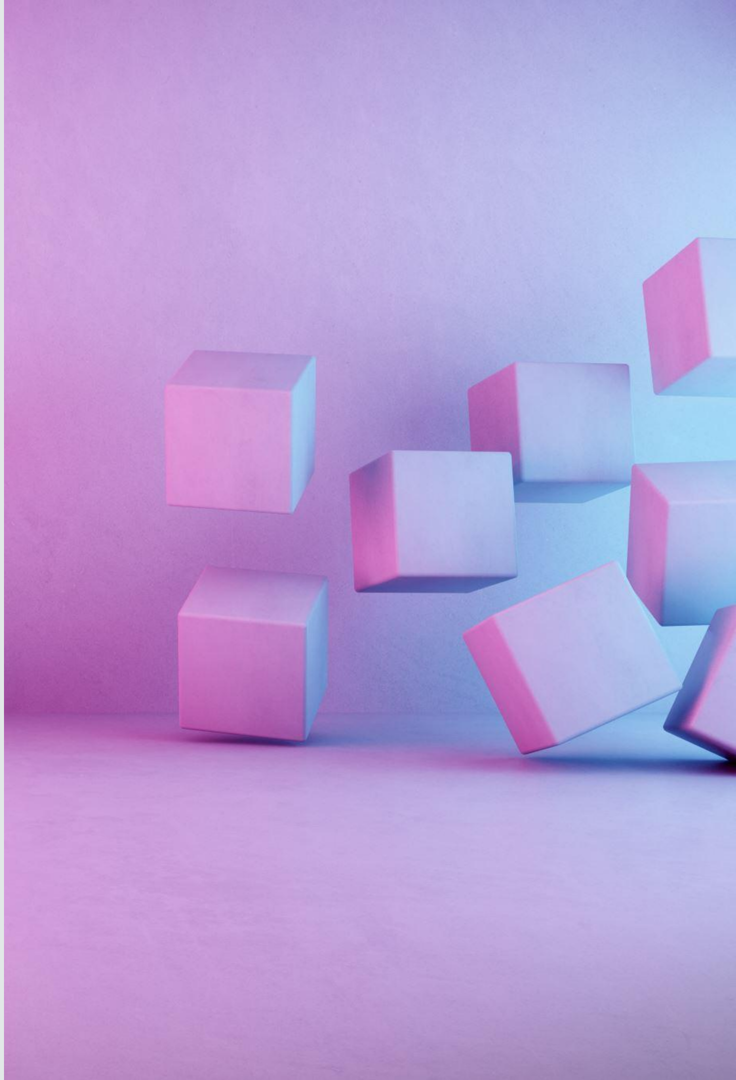
Implementing AdaBoost for Predicting Online Shoppers' Purchase Intentions

Team: 404 Not Found

Diksha Krishnan, Xinyu Zhou, Shen Yu, Dongyan Sun

December 9, 2024



How does it work?

- Weak Learners
- Weighting
- Iteration T
- Prediction

Pseudo Code

Input:

$S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, where $y_i \in \{-1, +1\}$,

number of iterations T,

weak learner WL

Initialize: Sample weights $D_i^{(1)} = \frac{1}{m}, \forall i = 1, \ldots, m$.

for $t = 1$ **to** T :

Invoke weak learner $h_t = WL(D^{(t)}, S)$

Compute error rate: $\epsilon_t = \sum_{i=1}^{m} D_i^{(t)} \mathbb{1}[h_t(x_i) \neq y_i]$.

Let $w_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$.

Update sample weights: $D_i^{(t+1)} = \frac{D_i^{(t)} \exp(-w_t y_i h_t(x_i))}{Z_t}$,

where $Z_t = \sum_{i=1}^{m} D_i^{(t)} \exp(-w_t y_i h_t(x_i))$, for all $i = 1, \ldots, m$

Output: Final hypothesis: $h_s(x) = \text{sign}\left(\sum_{t=1}^{T} w_t h_t(x)\right)$.

Advantages

- Simplicity and Flexibility
- Non-Linear Problems
- Resistant to Overfitting

Disadvantages

- Sensitive
- Computational Complexity
- Dependence on Weak Learners

Math Behind AdaBoost

Representation

1. The weak learner is Decision Stumps
2. A weighted sum of the predictions from all weak learners
3. Smaller error leads to a larger weight, indicating a better learner

Loss Function

1. Minimizes an exponential loss function which ensures that large errors (misclassified points) contribute more significantly to the loss
2. Encouraging the model to focus on correcting them

Optimizer

Equations of AdaBoost

$x \in \mathbb{R}^d$ and $y \in \{1, -1\}$.

$$H(x) = \text{sign}\left(\sum_{m=1}^{M} w_m h_m(x)\right)$$

$$w_m = \frac{1}{2}\ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$$

$$\mathcal{L}(H) = \sum_{i=1}^{m} \exp(-y_i H(x_i))$$

Math Behind AdaBoost

$H(x) = \operatorname{sign}\left(\sum_{m=1}^{M} w_m h_m(x)\right)$

Representation

$x \in \mathbb{R}^d$ and $y \in \{1, -1\}$.

Optimizer

$D_{t+1}(i) = D_t(i) \cdot \exp\left(-\alpha_t y_i h_t(x_i)\right)$

$$Z_t = \frac{1}{m} \sum_{i=1}^{m} e^{-y_i f_t(x_i)}.$$

$$w_m = \frac{1}{2} \ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$$

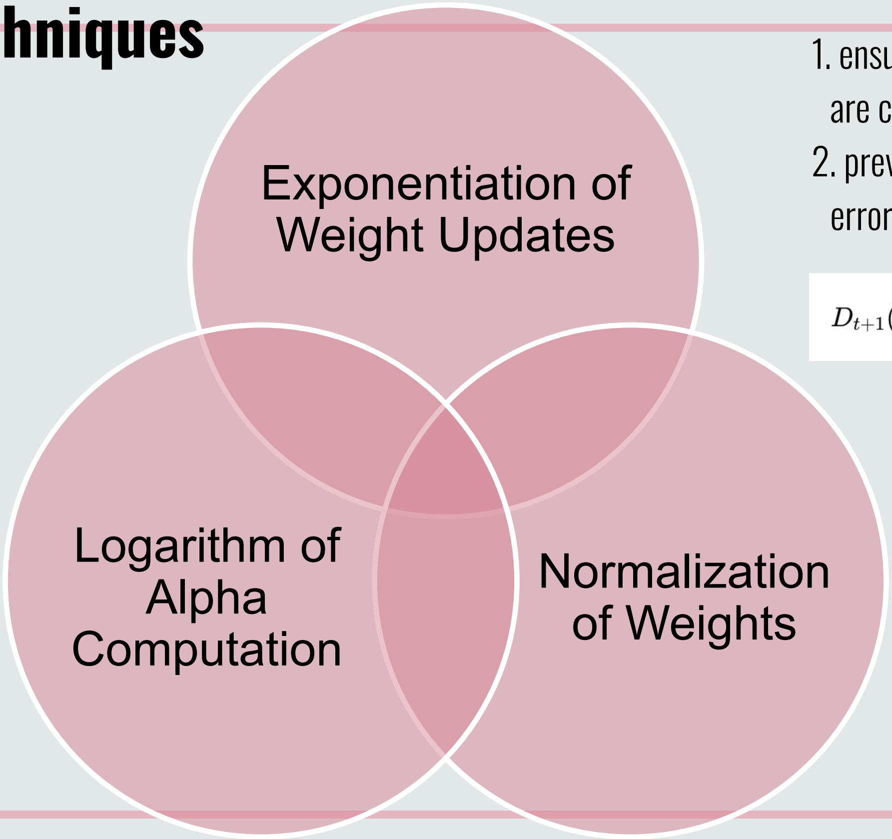
Loss

$\mathcal{L}(H) = \sum_{i=1}^{m} \exp(-y_i H(x_i))$

Numerical Techniques

$$w_m = \frac{1}{2}\ln\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$$

- ensures weak classifiers with low error receive higher weights
- introduces stability, by avoiding extremely large or small values of alpha



1. ensures that the weight updates are correctly scaled
2. prevent underflow or overflow errors

$$D_{t+1}(i) = D_t(i) \cdot \exp\left(-\alpha_t y_i h_t(x_i)\right)$$

1. after updating the weights, they are normalized
2. prevent the weights from growing too large, which could lead to numerical instability

Previous Work We Reproduced

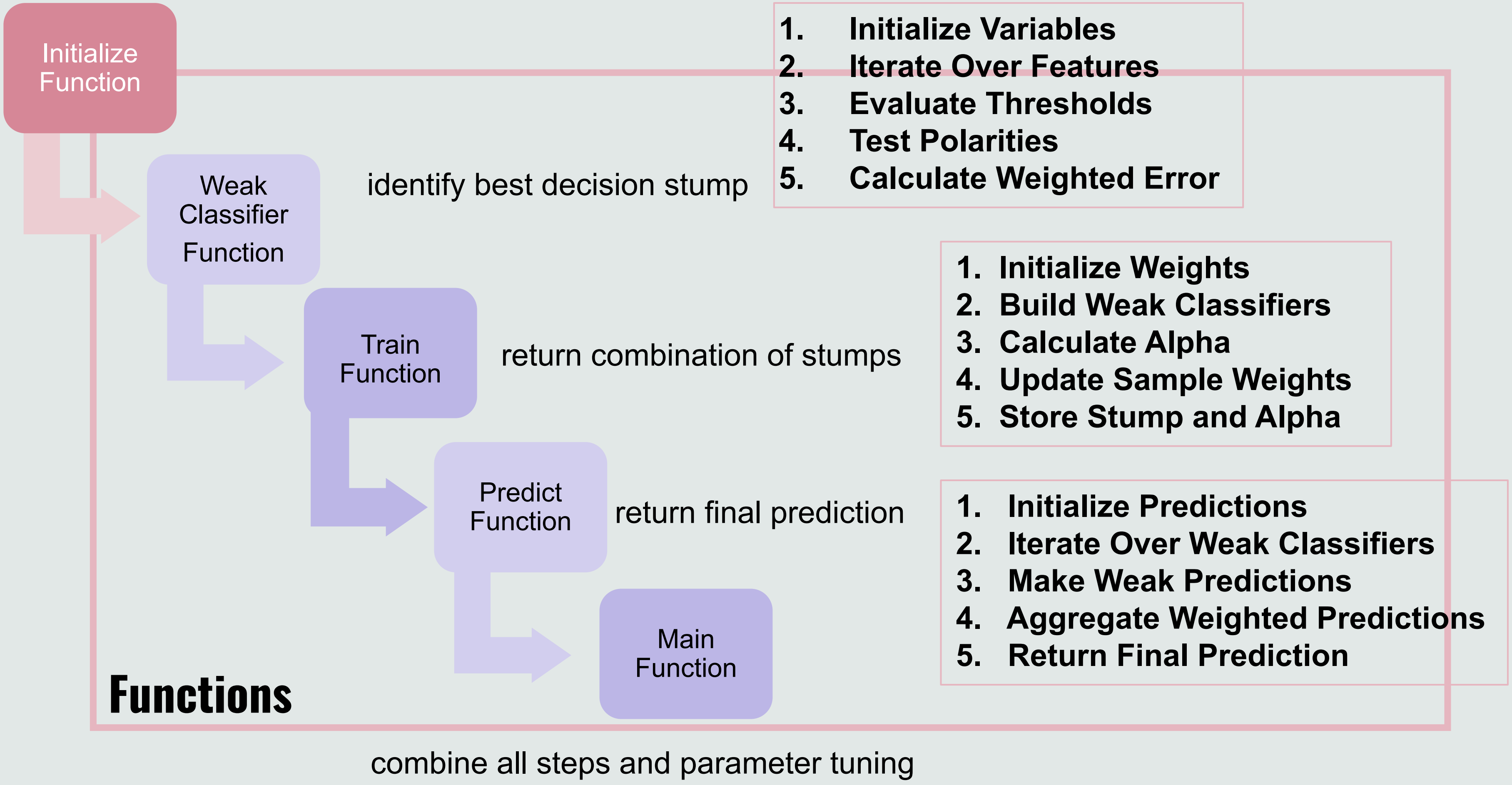
Target Variable → Y: which online shoppers will end with shopping → {1, -1} → Binary Classification

Feature → X: URL information of the pages visited → 10 continuous → 7 categorical

Preprocess → One Hot Encoder → Standard Scaler

Imbalance → Stratified Split → 60% Train → 20% Validation → 20% Test

Main Function to Combine All → Preprocess + Split → Initialize Function → Weak Classifier Function → Train Function → Predict Function


Initialize Function
Weak Classifier Function
identify best decision stump
1. Initialize Variables
2. Iterate Over Features
3. Evaluate Thresholds
4. Test Polarities
5. Calculate Weighted Error
Train Function
return combination of stumps
1. Initialize Weights
2. Build Weak Classifiers
3. Calculate Alpha
4. Update Sample Weights
5. Store Stump and Alpha
Predict Function
return final prediction
1. Initialize Predictions
2. Iterate Over Weak Classifiers
3. Make Weak Predictions
4. Aggregate Weighted Predictions
5. Return Final Prediction
Main Function
Functions
combine all steps and parameter tuning

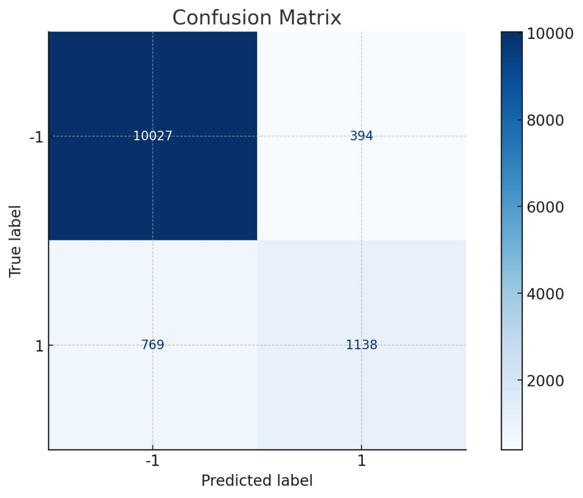
Result

Metric	AdaBoost Model	Previous Work
Accuracy	90.56%	89.14%
F1 Score	66.18%	63.26%
Precision	74.27%	69.34%
Recall	59.69%	58.17%

Slightly Higher Accuracy & F1 Score

Low Recall reduces Low F1 Score

Poor to Capture True Positives

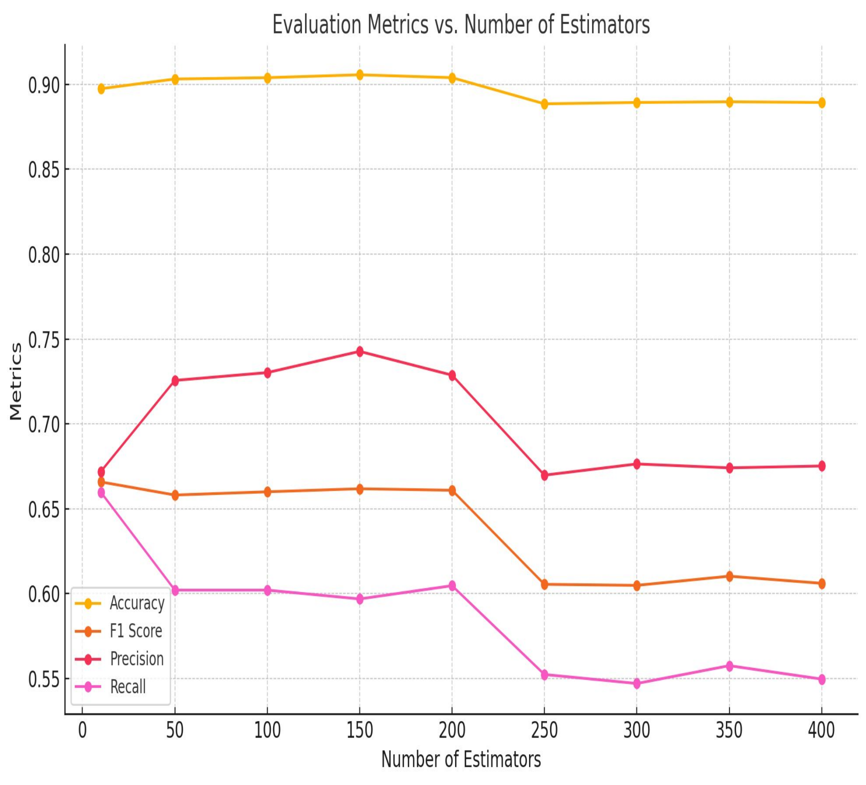


Action 1: Oversampling or Undersampling

Goal 1 : Improve Balance

Action 2: Replace Weak Classifier with Decision Tree

Goal 2: More Sensitive to Minority Class



Result

Tune the Number of Estimators

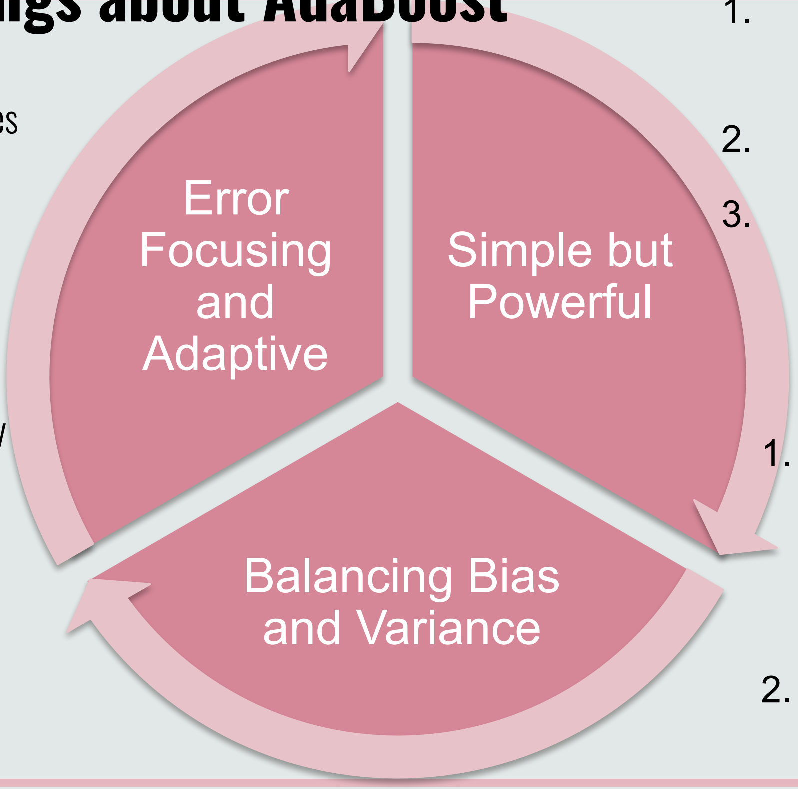
150: Best Performance

Above 200: Overfitting

Trade-off between Bias and Variance

Interesting Things about AdaBoost

1. prioritizes correcting mistakes made by earlier weak learners
2. iteratively updating weights for misclassified samples
3. focus on hard-to-classify points, making the model highly adaptive



1. combines weak learners (like decision stumps) into a strong ensemble
2. easy to implement
3. works effectively for both classification and regression problems

1. in the early stages, when bias is high, it focuses on improving the model by giving more weight to the misclassified examples
2. as the model improves, focus shifts toward balancing bias reduction with variance control

Challenges when implemented AdaBoost

Sensitivity to Noise

1. misclassified samples are given higher weights
2. if the misclassified samples are just noises or outliers, it can lead to poor performance
3. carefully preprocess the data

Computational Intensity

1. requires sequential training of multiple weak learners
2. slow when dealing with large datasets or complex base learners
3. this sequential nature also makes it less parallelizable

Complex Parameter Tuning

1. requires careful tuning of hyperparameters, such as the number of iterations and learning rate
2. incorrect tuning may lead to underfitting or overfitting
3. if the weak learners are too complex, the model may overfit the training data

Q&A

References:

1. Shalev-Shwartz, S. and Ben-David, S. (2014) Understanding Machine Learning: From Theory to Algorithms. Cambridge: Cambridge University Press.

2. Swetha, T., R, R., Sajitha, T., B, V., Sravani, J. and Praveen, B. (2024) 'Forecasting Online Shoppers Purchase Intentions with Cat Boost Classifier', 2024 International Conference on Distributed Computing and Optimization Techniques (ICDCOT), Bengaluru, India, 2024, pp. 1-6. doi: 10.1109/ICDCOT61034.2024.10515309.

Githubs: