AdaBoost - Quoted directly from the main report.

Joel Sundin

January 14, 2025

0.1 Adaptive Boosting (AdaBoost)

0.1.1 Mathematical Model

Boosting is an ensemble learning technique where multiple weak classifiers are combined in order to form a strong classifier. In the case of Adaboost (Adaptive boosting), the weak learners are typically decision stumps, which are shallow decision trees of low depth. Although each decision stump is simple and might fail to capture the complexity of the dataset individually, their combination through boosting results in a powerful classifier.

For a training dataset $\mathcal{T} = \{(x_i, y_i)\}_{i=1}^N$, where x_i are the input features and y_i the target labels, AdaBoost adjusts the importance of each sample during training. Misclassified samples receive higher weights, forcing the next weak learner to focus on these harder-to-classify points.

The final prediction $\hat{y}$ is obtained by taking a weighted majority vote across all weak classifiers, where the weights reflect each weak learner's performance.

Weak learners Decision stumps The weak learners partition the data based on a single feature and a threshold. For each possible split in the training data $\mathcal{T}$, the algorithm evaluates conditions in the form:

$$\{x_j \leq s_k\} \text{ } or \text{ } \{x_j > s_k\}$$

where x_j is the feature being split and s_k is the threshold value.

In order to find good splits, the algorithm evaluates possible splits between all unique values of each feature. The goal of the fitting process is to find the split that minimizes an impurity measure $Q(s)$, which quantifies the *impurity* or *disorder* at a node. Common impurity measures include **Gini impurity** and **Entropy**. The chosen split (x_j, s_k) is the one that minimizes the impurity:

$$s_k = \arg\min_s Q(s)$$

Weighting of a weak learner Once the best split has been found, the performance of the weak learner is evaluated by calculating its *weighted classification error*. This weighted error, ϵ_t, is computed as the sum of the sample weights of the misclassified examples:

$$\epsilon_t = \frac{\sum_{i=1}^N w_i \cdot \mathbb{I}(h_t(x_i) \neq y_i)}{\sum_{i=1}^N w_i}$$

where w_i is the weight of the i-th datapoint (sample). This error rate is then used to compute the *weight of the weak learner* for the final prediction. If the weak learner has a low error rate (i.e., ϵ_t is small), it is assigned a higher weight in the final prediction. The weight α_t for the weak learner is calculated as:

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

This weight determines how much influence the weak learner will have on the final prediction. The weighted sum of all weak learners' predictions is used to make the final classification, with the weight α_t ensuring that better-performing trees contribute more to the final decision.

This weighting allows for AdaBoost to focus more on the *harder* samples in subsequent iterations, as the misclassified samples' weights are increased, forcing the next weak learner to pay more attention to those examples. **Training Adaboost** The iterative process of training the Adaboost consists of multiple steps:

1. Initialize weights $\{w_i\}_{i=1}^{N}$
2. For T weak learners:
   (a) Train weak learner
   (b) Compute the weighted error ϵ_t for the weak learner
   (c) Update the weight α of the weak learner, based on the weighted error
   (d) Update the weights of the training samples $w_i = w_i * \exp(\alpha * \mathbb{I}(h(x_i) \neq y_i))$

0.1.2 Implementation

To implement the AdaBoost algorithm, we used the `AdaBoostClassifier` from the `sklearn.ensemble` library, which provides a user-friendly implementation of the AdaBoost method for classification tasks. This implementation allows us to experiment with hyperparameters such as the *base_ classifier*, *n_ estimators* and *learning_ rate* (being a float used in the model weighting process, controlling the intensity of the weighting). The ability to quickly swap out base classifiers enables us to explore different options for the weak learners, such as varying the depth of the decision trees or using different impurity measures.

The model was tuned using `GridSearchCV`, where the performance while alterating following hyperparameters where examined:

- DecisionTreeClassifier (base_estimator)
  - criterions: ['gini', 'entropy', 'log_loss']
  - depth: [1, 2, 3, 4, 5, 6, 7]
- n_estimators: [50 $\rightarrow$ 400 (*in steps of 25*)]
- learning_rate: [0.01, 0.1, 1.0, 1.5]

0.1.3 Results

The best-performing models, found according to the algorithm from the previous section, are selected based on their F1-score. This metric is preferred because the test partion of the dataset remains imbalanced, making other metrics, such as accuracy, less effective at reflecting the true class distributions.

Although the high complexity of this model raises potential concerns about overfitting, its performance on unseen test data indicates good generalization capabilities. The model achieves satisfactory results in identifying the non-majority class, striking an appropriate balance between precision and recall despite the challenges posed by class imbalance.

Since the nature of the data differs due to the PCA transformation, we extract two distinct models—one for each dataset, presented in Table 1. The achieved performance metrics of these models is summarized in Table 2.

Table 1: Adaboost hyperparameters

Model	**Non-PCA**	**PCA**
Criterion	gini	gini
depth	5	3
estimators	200	375
learning rate	1	1

Table 2: Adaboost performance evaluation

Metric	**Non-PCA**	**PCA**
F1 Score	0.634	0.578
Accuracy	0.906	0.881
Precision	0.722	0.591
MSE	0.094	0.12

1 Conclusion

Based on the results of the multiple performance evaluations presented in Table 3, Adaboost emerged as the most effective model, outperforming others in terms of both accuracy and F1-score. While kNN demonstrated a marginally higher precision, Adaboost consistently delivered superior results across multiple evaluation metrics, making it the more robust choice overall. Adaboost achieved the highest F1-score at 63.4%, significantly surpassing kNN's score of 47% and Logistic Regression's score of 38%. These findings suggest that Adaboost is particularly well-suited to datasets with larger feature spaces, as it effectively captures non-linear relationships and complex patterns that are challenging for simpler models to handle.

Given its demonstrated effectiveness across the presented data compared to the other models, Adaboost has been selected as the final model for deployment on the provided test dataset.

Table 3: Performance Evaluation of Models

Best Parameters (Adaboost)	Estimator: *DecisionTreeClassifier*(max_depth=5), *learning rate*: 1.0, *n estimators*: 200
Adaboost	• Accuracy: 90.6% • Precision: 72.2% • F1 Score: 63.4%
Logistic Regression	• Accuracy: 70% • Precision: 27% • F1 Score: 38%
kNN	• Accuracy: 89% • Precision: 73% • F1 Score: 47%