

NYCU Introduction to Machine Learning, Homework 3

112550077, 劉逢穎

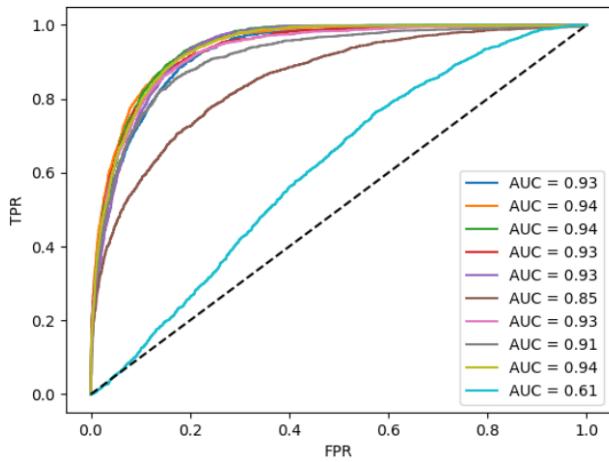
Part. 1, Coding (60%):

(20%) Adaboost

1. (5%) Show your accuracy of the testing data (`n_estimators = 10`)

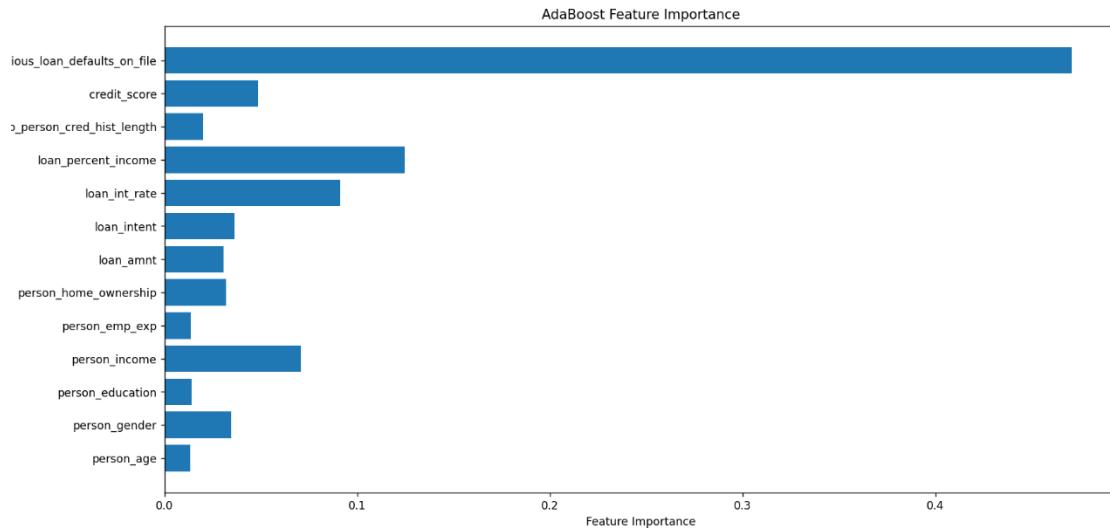
```
2025-11-12 14:20:35.438 | INFO      | __main__:main:45 - AdaBoost - Accuracy: 0.8859
```

2. (5%) Plot the AUC curves of each weak classifier.



3. (10%)

- Plot the feature importance of the AdaBoost method.



- Paste the snapshot of your implementation of the feature importance estimation.

```

def compute_feature_importance(self) -> t.Sequence[float]:
    """
    TODO: Implement the feature importance calculation
    """
    num_features = self.learners[0].fc1.in_features
    feature_importance = np.zeros(num_features)

    for t, model in enumerate(self.learners):
        weights = model.fc1.weight.detach().numpy().flatten()
        feature_importance += self.alphas[t] * np.abs(weights)

    feature_importance /= np.sum(feature_importance)

    return feature_importance.tolist()
    raise NotImplementedError

```

- c. Explain how you compute the feature importance for the Adaboost method shortly (within 100 words as possible)

Ans:

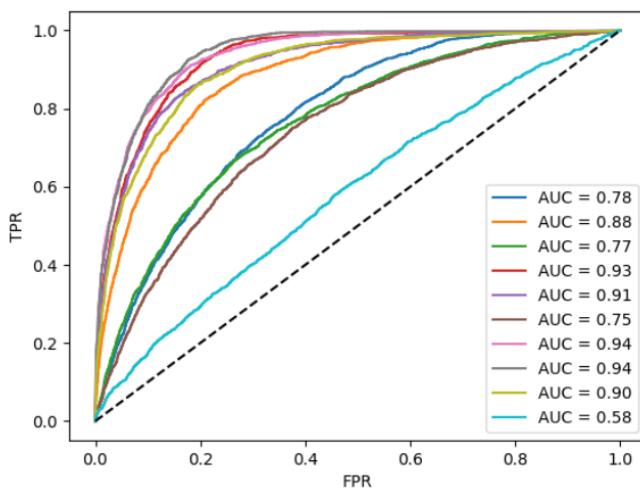
I combine the absolute feature weights from each weak learner, weighted by the learner's α value (its contribution to the ensemble). Features with consistently high absolute weight across strong learners receive higher importance. Unlike Bagging, which treats all learners equally. This is to say that its importance reflects both feature influence and learner confidence.

(20%) Bagging

4. (5%) Show the accuracy of the test data using 10 estimators. (n_estimators=10)

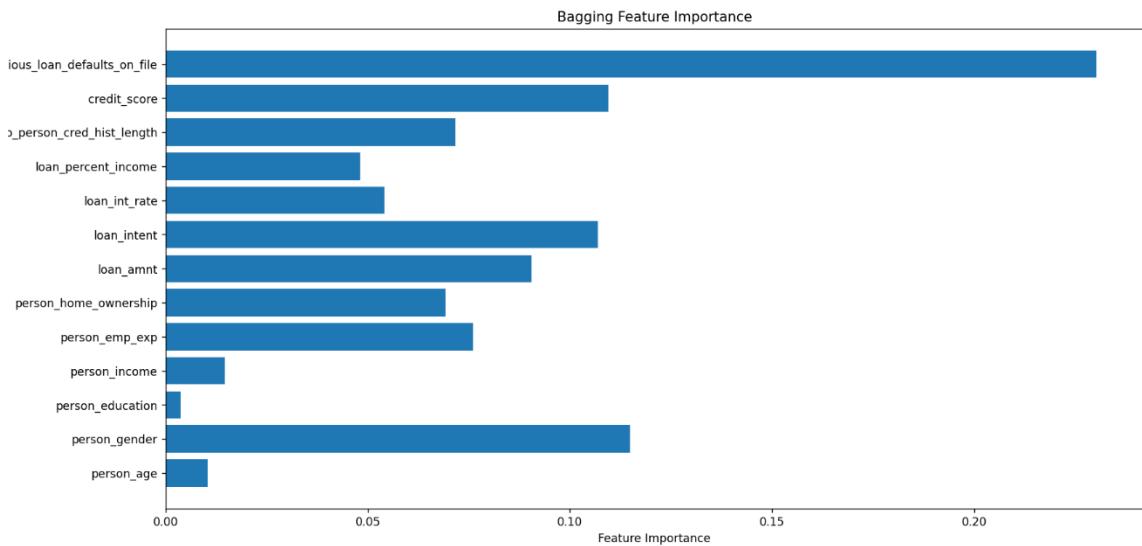
2025-11-12 14:27:58.765 | INFO | __main__:main:63 - Bagging - Accuracy: 0.8517

5. (5%) Plot the AUC curves of each weak classifier.



6. (10%)

- a. Plot the feature importance of the Bagging method.



- b. Paste the snapshot of your implementation of the feature importance estimation.

```
def compute_feature_importance(self) -> t.Sequence[float]:
    """
    TODO: Implement the feature importance calculation
    """
    num_features = self.learners[0].fc1.in_features
    feature_importance = np.zeros(num_features)

    for t, model in enumerate(self.learners):
        feature_importance += np.abs(model.fc1.weight.detach().numpy().flatten())

    feature_importance /= np.sum(feature_importance)

    return feature_importance.tolist()
```

- c. Explain how you compute the feature importance for the Bagging method shortly (within 100 words as possible)

Ans:

I average the absolute feature weights from all weak learners, treating each learner equally since they are trained independently on bootstrap samples. This measures how strongly each feature contributes across the ensemble. Unlike AdaBoost, Bagging does not weight learners by performance (no α values)

(15%) Decision Tree

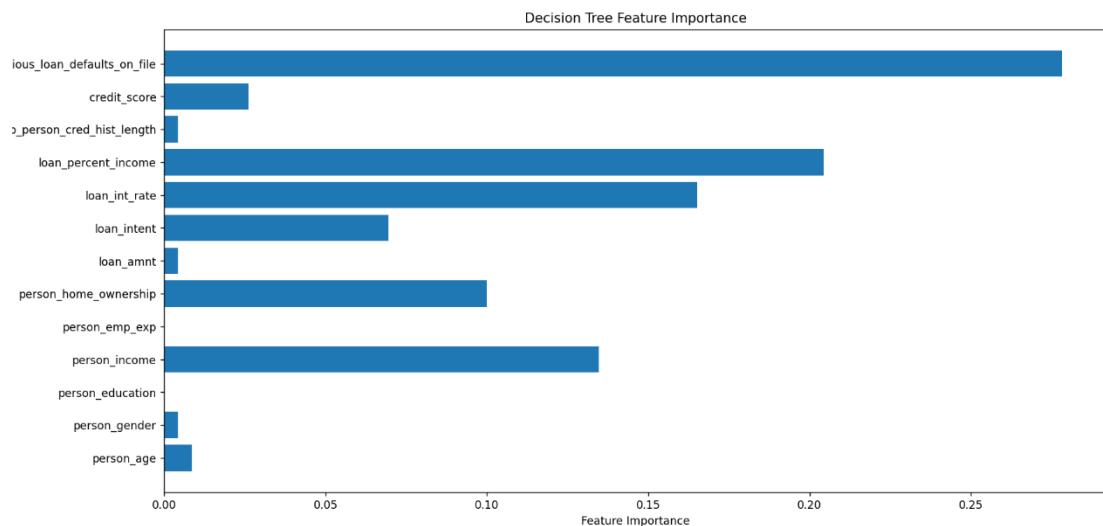
7. (5%) Compute the Gini index and the entropy of the array [0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1].

```
2025-11-12 14:25:12.020 | INFO | __main__:main:84 - DecisionTree - Gini index: 0.4959
2025-11-12 14:25:12.020 | INFO | __main__:main:89 - DecisionTree - entropy: 0.9940
```

8. (5%) Show your accuracy of the testing data with a max-depth = 7

```
2025-11-12 14:25:12.020 | INFO | __main__:main:79 - DecisionTree - Accuracy: 0.9089
```

9. (5%) Plot the feature importance of the decision tree.



(5%) Code Linting

10. Show the snapshot of the flake8 linting result (paste the execution command even when there is no error).

- PS D:\ML\114-1-ml-hw3-release> **flake8 main.py**
- PS D:\ML\114-1-ml-hw3-release> []

Part. 2, Questions (40%):

1. (15%)

In the AdaBoost algorithm, each selected weak classifier, h_t is assigned a weight:

$$\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$$

- (a) What is the definition of ϵ_t in this formula?
- (b) If a weak classifier h_t performs only as well as "random guessing" (i.e., $\epsilon_t = 0.5$), what will α_t be?
- (c) Based on your answer in (b), briefly explain how this α_t weight formula ensures the robustness of AdaBoost.

Ans:

- (a) ϵ_t is the weighted error rate of the weak classifier h_t , computed as the sum of the sample weights of all misclassified examples.
- (b) If $\epsilon_t = 0.5$, then $\alpha_t = 0$, meaning the weak classifier contributes nothing to the final ensemble.
- (c) When $\epsilon_t < 0.5$, α_t becomes positive, giving more influence to good classifiers. If $\epsilon_t > 0.5$, α_t would be negative, which reduces or reverses its effect. This weighting mechanism ensures that weak classifiers performing better than random guessing have more impact, making AdaBoost robust against poor learners.

2. (15%) Random Forest is often considered an improvement over Bagging (Bootstrap Aggregating) when used with Decision Trees as the base learners.

- (a) Briefly explain the potential problem that can exist among the T classifiers (C_1, \dots, C_T) produced by Bagging when using decision trees.
- (b) To mitigate the problem from (a), Random Forest introduces a second source of randomness in addition to Bagging. Specifically describe what this second source of randomness is and how it works.

Ans:

- (a) When using decision trees as base learners, Bagging may produce highly correlated trees because each tree can select the same strong features at the top splits. This correlation reduces the variance reduction effect of Bagging.

(b) Random Forest introduces an additional source of randomness by selecting a random subset of features at each split. This means each tree considers only a random subset of attributes when deciding how to split, which reduces correlation between trees and improves generalization.

3. (10%) Is it always possible to find the smallest decision tree that codes a dataset with no error in polynomial time? If not, what algorithm do we usually use to search a decision tree in a reasonable time? Also, list the criterion that we stop splitting the decision tree and leaf nodes are created (List two).

Ans:

No. Finding the smallest decision tree consistent with the training data is an NP-complete problem, so it cannot be solved in polynomial time. We usually use greedy algorithms such as ID3 or CART(based on gini or information gain) to construct a reasonably good tree.

1. All samples in a node belong to the same class (pure node).
2. A stopping criterion (like maximum depth or minimum samples per leaf) is met.