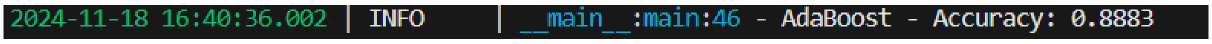
NYCU Introduction to Machine Learning, Homework 3

[111550149], [林悦揚]

Part. 1, Coding (60%):

(20%) Adaboost

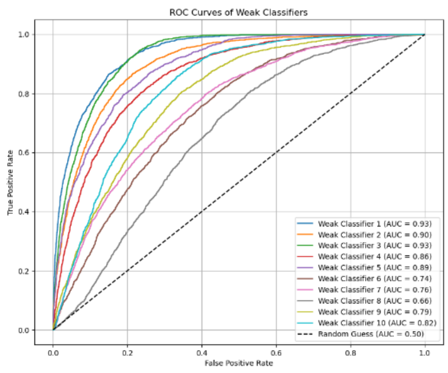
1. (10%) Show your accuracy of the testing data (n_estimators = 10)



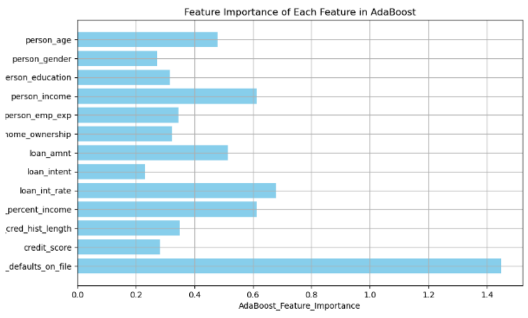
```
2024-11-18 16:40:36.002 | INFO     | __main__:main:46 - AdaBoost - Accuracy: 0.8883
```

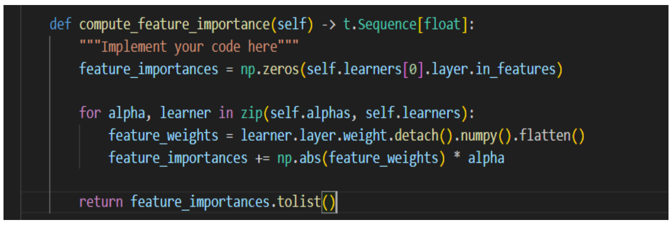
num_epochs=500, learning_rate=0.007

2. (5%) Plot the AUC curves of <u>each</u> weak classifier.



3. (5%) Plot the feature importance of the AdaBoost method. Also, you should snapshot the implementation to calculate the feature importance.



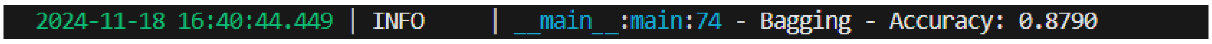


```python
def compute_feature_importance(self) -> t.Sequence[float]:
    """Implement your code here"""
    feature_importances = np.zeros(self.learners[0].layer.in_features)

    for alpha, learner in zip(self.alphas, self.learners):
        feature_weights = learner.layer.weight.detach().numpy().flatten()
        feature_importances += np.abs(feature_weights) * alpha

    return feature_importances.tolist()
```

This function calculates the importance of each feature in the model by summing the absolute values of feature weights from all weak classifiers, weighted by their contribution coefficients (alpha), to assess the overall impact of each feature on the model.

(20%) Bagging

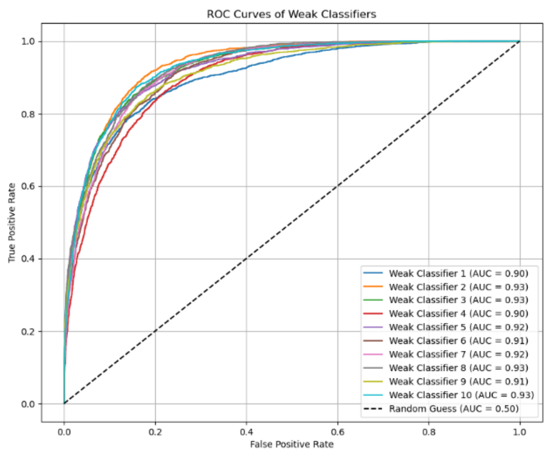
4. (10%) Show your accuracy of the testing data with 10 estimators. (n_estimators=10)



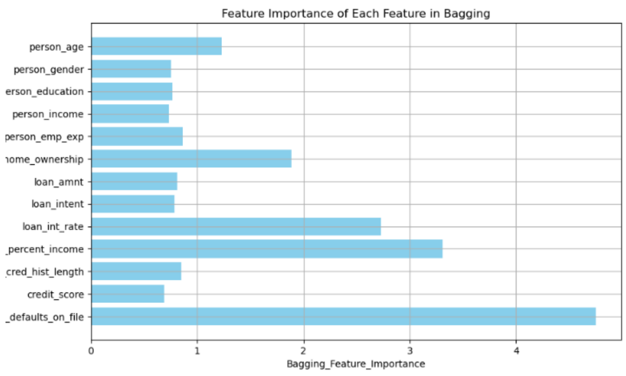
```
2024-11-18 16:40:44.449 | INFO     | __main__:main:74 - Bagging - Accuracy: 0.8790
```

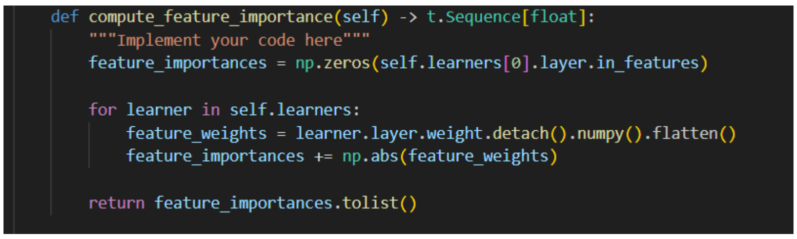
num_epochs=500, learning_rate=0.007

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



6. (5%) Plot the feature importance of the Bagging method. Also, you should snapshot the implementation to calculate the feature importance.



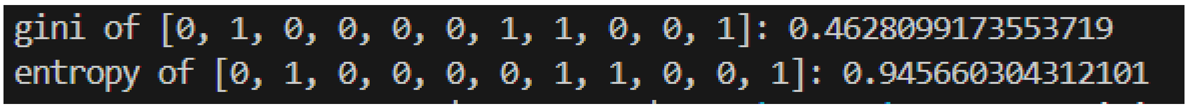


```python
    def compute_feature_importance(self) -> t.Sequence[float]:
        """Implement your code here"""
        feature_importances = np.zeros(self.learners[0].layer.in_features)

        for learner in self.learners:
            feature_weights = learner.layer.weight.detach().numpy().flatten()
            feature_importances += np.abs(feature_weights)

        return feature_importances.tolist()
```

This function calculates feature importance by summing the absolute values of feature weights from all weak classifiers, returning a list that reflects the overall contribution of each feature to the model.

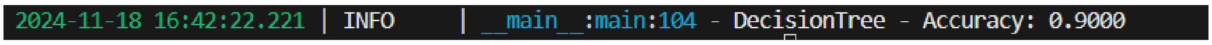
(15%) Decision Tree

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



```
gini of [0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1]: 0.4628099173553719
entropy of [0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1]: 0.945660304312101
```

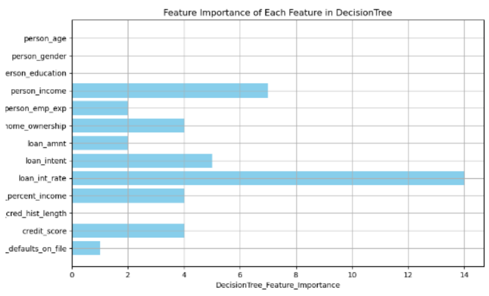
8. (5%) Show your accuracy of the testing data with a max-depth = 7

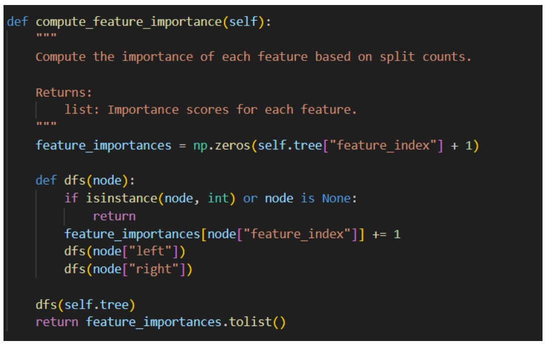


```
2024-11-18 16:42:22.221 | INFO     | __main__:main:104 - DecisionTree - Accuracy: 0.9000
```

Selecting 80% of the original data with random_state=45 to avoid overfitting.Because the random_state is constant , this model produce the same results when rebuilt with the same arguments.

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



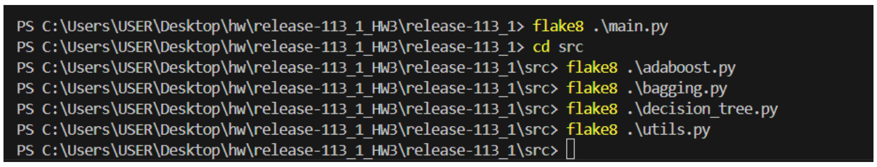


```python
def compute_feature_importance(self):
    """
    Compute the importance of each feature based on split counts.

    Returns:
        list: Importance scores for each feature.
    """
    feature_importances = np.zeros(self.tree["feature_index"] + 1)

    def dfs(node):
        if isinstance(node, int) or node is None:
            return
        feature_importances[node["feature_index"]] += 1
        dfs(node["left"])
        dfs(node["right"])

    dfs(self.tree)
    return feature_importances.tolist()
```

This function computes the importance of each feature in a decision tree by recursively traversing all nodes in the tree, counting how many times each feature is used as a splitting node. It returns a list where the importance of each feature is represented by the number of times it is used, with higher counts indicating greater contributions to the model.

(5%) Code Linting

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



```
PS C:\Users\USER\Desktop\hw\release-113_1_HW3\release-113_1> flake8 .\main.py
PS C:\Users\USER\Desktop\hw\release-113_1_HW3\release-113_1> cd src
PS C:\Users\USER\Desktop\hw\release-113_1_HW3\release-113_1\src> flake8 .\adaboost.py
PS C:\Users\USER\Desktop\hw\release-113_1_HW3\release-113_1\src> flake8 .\bagging.py
PS C:\Users\USER\Desktop\hw\release-113_1_HW3\release-113_1\src> flake8 .\decision_tree.py
PS C:\Users\USER\Desktop\hw\release-113_1_HW3\release-113_1\src> flake8 .\utils.py
PS C:\Users\USER\Desktop\hw\release-113_1_HW3\release-113_1\src> 
```

Part. 2, Questions (40%):

1. (10%) What are Bagging and Boosting, and how are they different? Please list their difference and compare the two methods in detail.

Bagging creates multiple subsets of the original dataset using random sampling with replacement, trains a weak model (e.g., decision tree) on each subset, and combines the results through averaging (for regression) or majority voting (for classification). It primarily reduces variance.

Boosting is an iterative technique where each weak model is trained sequentially. Data points are weighted based on the errors of the previous model, and the final predictions are made by combining all weak models with weights. It primarily reduces bias.

	Bagging	Boosting
Sample Selection	With replacement, each selection is independent	Adjusted based on the previous model's performance
Sample Weighting	Equal	Higher weight given to samples with higher classification error
Model Weighting	Equal	Higher weight given to models with lower error rates
Parallel Processing	Possible	Not possible
Purpose	Reduces Variance	Reduces Bias

2. (15%) What criterion do we use to decide when we stop splitting while performing the decision tree algorithm? Please list at least three criteria.

   1. The information gain is less than a threshold: Stop splitting when the information gain (or another metric, such as Gini index) falls below a certain threshold.
   2. The minimum number of data per leaf: The minimum number of samples required to be at a leaf node.
   3. Maximum tree depth is reached: Set a maximum depth for the decision tree to prevent overfitting.

3. (15%) A student claims to have a brilliant idea to make random forests more powerful: since random forests prefer trees which are diverse, i.e., not strongly correlated, the student proposes setting m = 1, where m is the number of random features used in each node of each decision tree. The student claims that this will improve accuracy while reducing variance. Do you agree with the student's claims? Clearly explain your answer.

   When m=1, each node in a decision tree considers only a single random feature for splitting. This can lead to highly correlated trees, which reduces the diversity among them and limits the advantages of ensemble learning. As a result, the model's predictions may have higher variance. Selecting a subset of features with m>1 allows the trees to explore different combinations of attributes, increasing diversity and enabling them to capture various patterns in the data. Therefore, using m>1 is generally recommended for achieving better accuracy and reducing prediction variance.