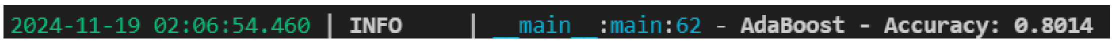
NYCU Introduction to Machine Learning, Homework 3

[111550108], [吳佳諭]

Part. 1, Coding (60%):

(20%) Adaboost

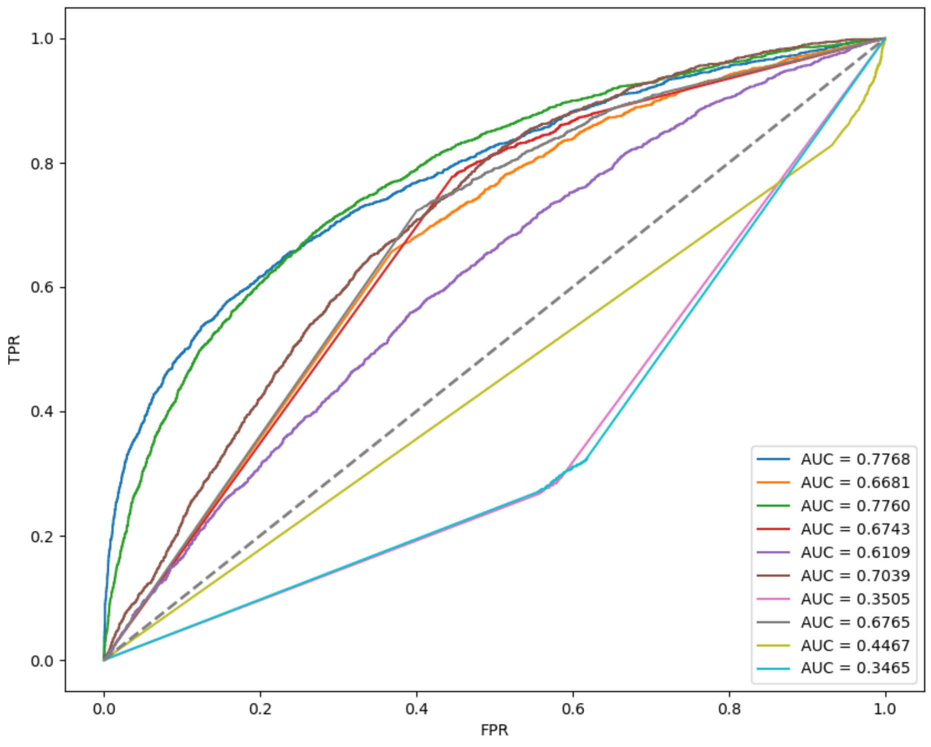
1. (10%) Show your accuracy of the testing data (n_estimators = 10)



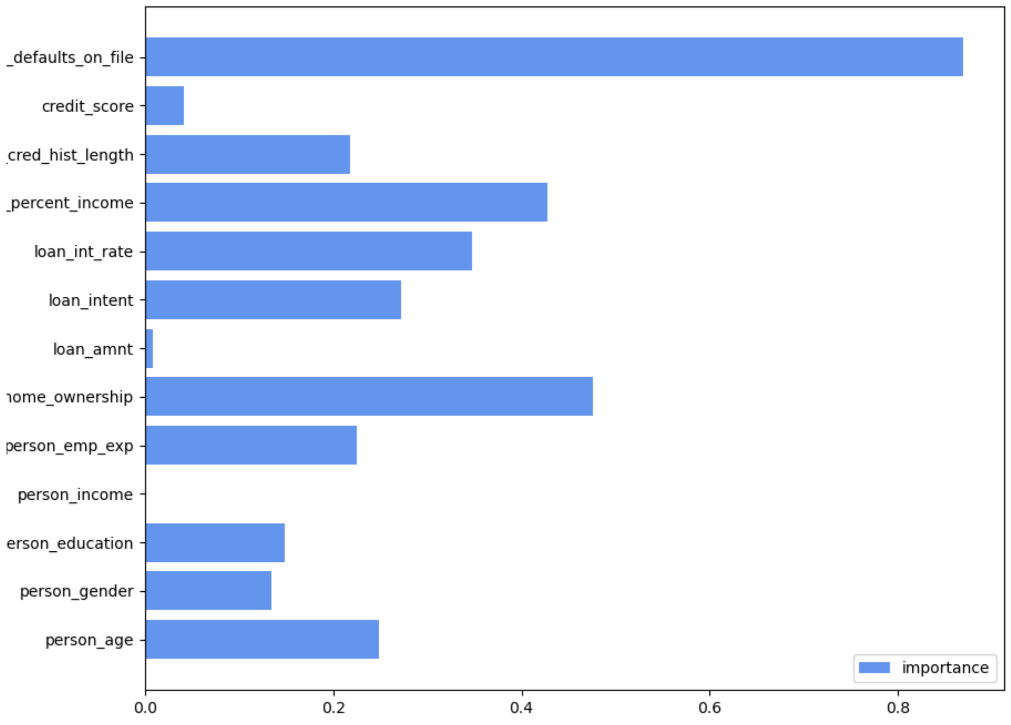
```
2024-11-19 02:06:54.460 | INFO     | __main__:main:62 - AdaBoost - Accuracy: 0.8014
```

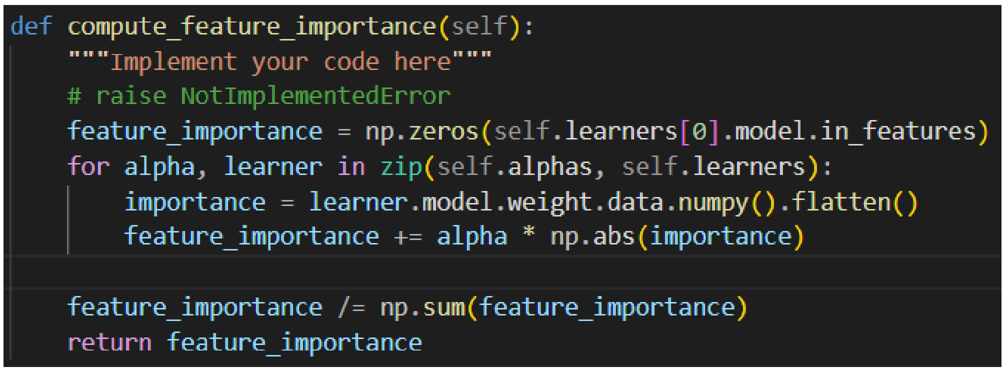
2. (5%) Plot the AUC curves of each weak classifier.

The last four learners have a low AUC (even lower than 0.5). It may be due to overfitting. However, even though some of the weak learners have a bad performance, the final performance of the strong classifier, which is the combination of the ten weak learners, can still has a good performance.



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



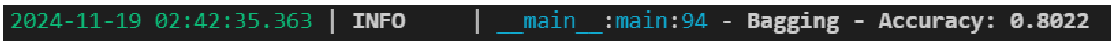


```
def compute_feature_importance(self):
    """Implement your code here"""
    # raise NotImplementedError
    feature_importance = np.zeros(self.learners[0].model.in_features)
    for alpha, learner in zip(self.alphas, self.learners):
        importance = learner.model.weight.data.numpy().flatten()
        feature_importance += alpha * np.abs(importance)

    feature_importance /= np.sum(feature_importance)
    return feature_importance
```

(20%) Bagging

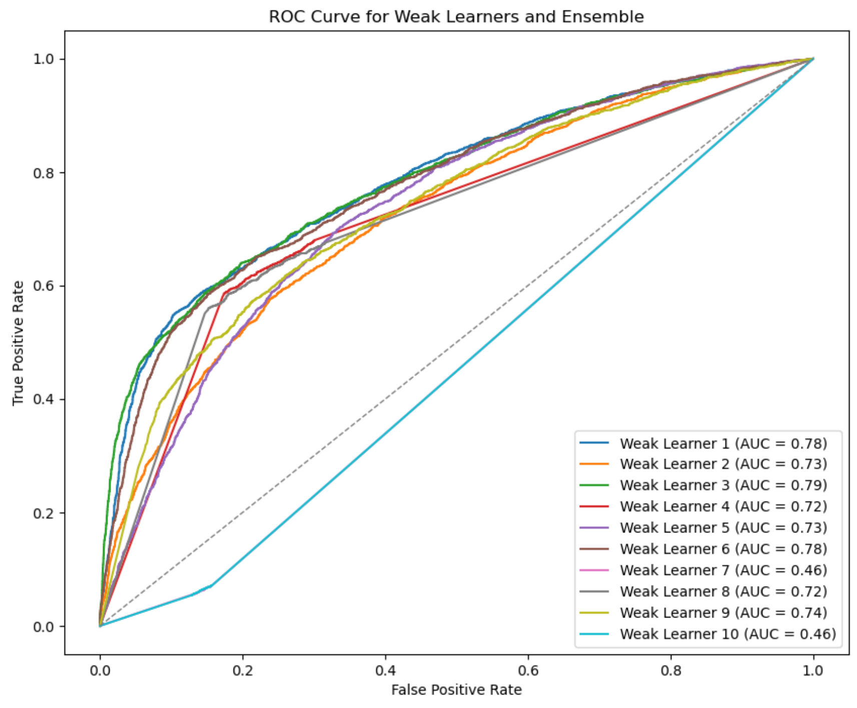
1. (10%) Show your accuracy of the testing data with 10 estimators. (n_estimators=10)



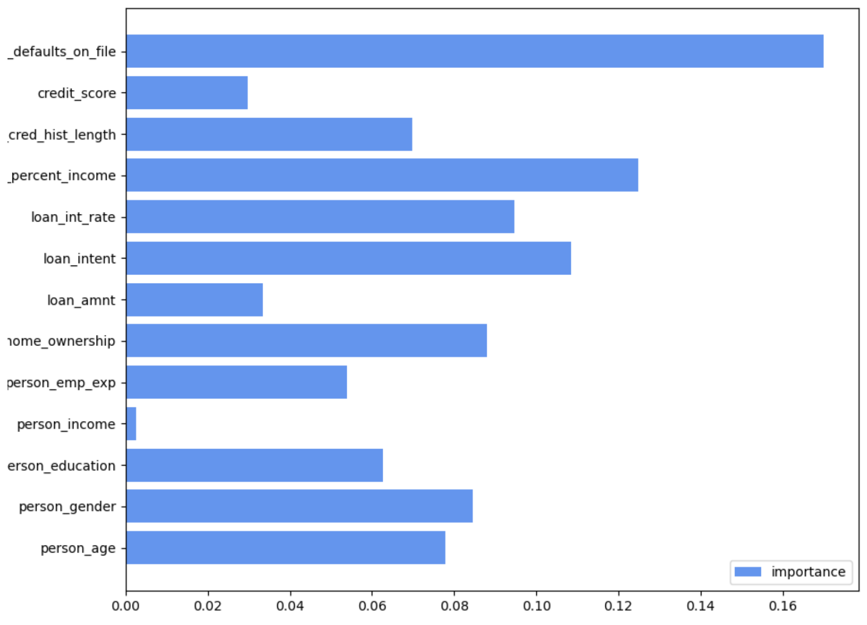
```
2024-11-19 02:42:35.363 | INFO     | __main__:main:94 - Bagging - Accuracy: 0.8022
```

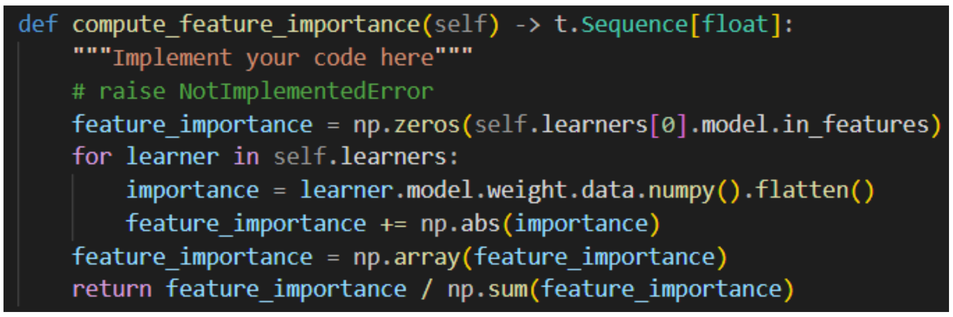
2. (5%) Plot the AUC curves of each weak classifier.

The last learner has a low AUC (0.46). It may be due to overfitting. However, even though one of the weak learners has a bad performance, the final performance of the strong classifier, which is the combination of the ten weak learners, can still has a good performance.



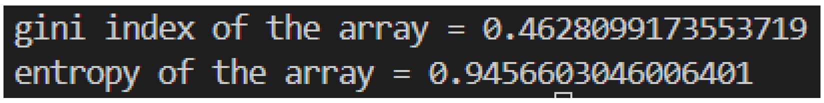
3. (5%) Plot the feature importance of the Bagging method. Also, you should snapshot the implementation to calculate the feature importance.




```
def compute_feature_importance(self) -> t.Sequence[float]:
    """Implement your code here"""
    # raise NotImplementedError
    feature_importance = np.zeros(self.learners[0].model.in_features)
    for learner in self.learners:
        importance = learner.model.weight.data.numpy().flatten()
        feature_importance += np.abs(importance)
    feature_importance = np.array(feature_importance)
    return feature_importance / np.sum(feature_importance)
```

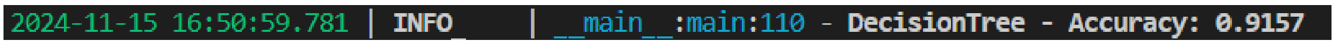
(15%) Decision Tree

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



```
gini index of the array = 0.4628099173553719
entropy of the array = 0.9456603046006401
```

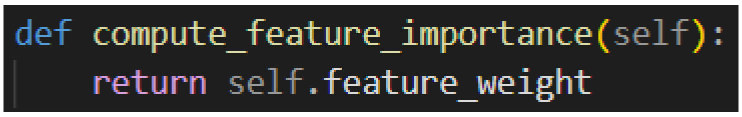
2. (5%) Show your accuracy of the testing data with a max-depth = 7



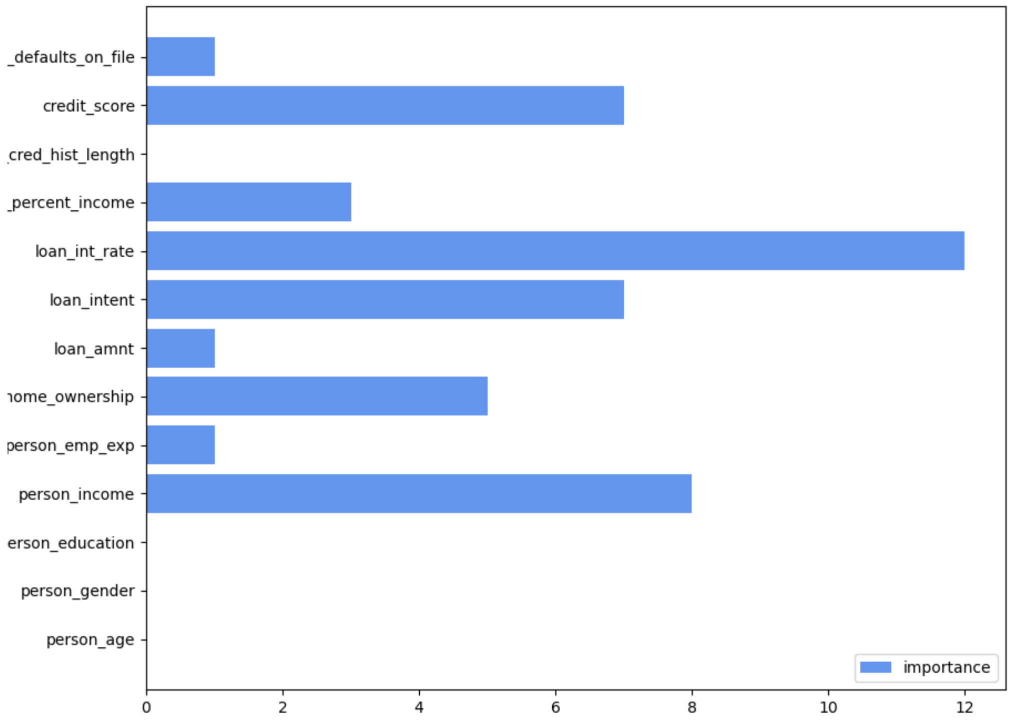
```
2024-11-15 16:50:59.781 | INFO     | __main__:main:110 - DecisionTree - Accuracy: 0.9157
```

At first, I split the data using threshold = X[:, feature_index], but this can only give an accuracy of 0.778, no matter how large I increase the depth. Then I use threshold = (threshold[i] + threshold[i-1]) / 2, the midpoint of the two threshold, then I get the accuracy of 0.9157!

3. (5%) Plot the feature importance of the decision tree.

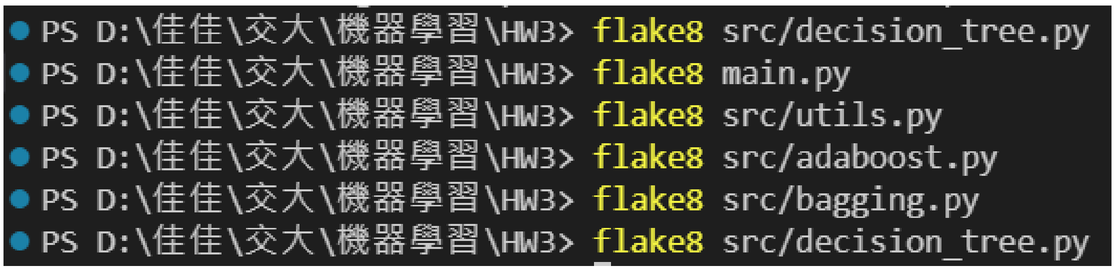


```
def compute_feature_importance(self):
    return self.feature_weight
```



(5%) Code Linting

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



```
● PS D:\佳佳\交大\機器學習\HW3> flake8 src/decision_tree.py
● PS D:\佳佳\交大\機器學習\HW3> flake8 main.py
● PS D:\佳佳\交大\機器學習\HW3> flake8 src/utils.py
● PS D:\佳佳\交大\機器學習\HW3> flake8 src/adaboost.py
● PS D:\佳佳\交大\機器學習\HW3> flake8 src/bagging.py
● PS D:\佳佳\交大\機器學習\HW3> flake8 src/decision_tree.py
```

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 and Boosting are both ensemble learning algorithms. They combine many weak learners to form a strong learner and use this strong learner to produce the final prediction. The weak learner only needs to perform better than random guessing. The final prediction is based on weighted vote or majority vote among the predictions made by the weak learners. Therefore, they are easy to implement and have good results.

Boosting

- The learners are trained sequentially. They will focus on the data that is misclassified by the previous learners by increasing the data weight. By doing this, they can correct the mistakes and minimize the error.

- While training, we have to calculate the error rate of the learner and maintain the data weight and the learner weight, such that the next learner can focus more on the misclassified data.

- The final prediction is made by the weighted vote from all the weak learners. That is, final_pred $=$ learner_pred $*$ learner_weight. If the learner has a lower error rate, it will be given a higher weight, and thus contribute more to the final prediction.

- Advantage: improve accuracy. Work well with weak learners.
  Disadvantage: may overfit due to the noise or outliers. Hard to parallelize.

Bagging

- The learners are trained independently. The dataset used to train the learner is sampled from the whole training set with a sampling with replacement strategy. By doing so, the learner can prevent overfitting.

- While training, learners only focus on the dataset they get. The algorithm is parallelizable because learners are trained independently.

- The final prediction is made by the majority vote from all the weak learners. That is, if over half of the learner says this data belongs to class 0, then the final prediction will be class 0.

- Advantage: reduce variance and overfitting (lower error). Parallelizable. Disadvantage: doesn't make weak learners strong. Time increase.

Comparison

Difference	Boosting	Bagging
Learners	Sequential	Independent
Train	Correct previous errors	Random data
Focus	Improve performance	Avoid overfitting
Prediction	Weighted vote	Majority vote
Advantage	Higher accuracy	Avoid overfitting
Disadvantage	Overfit	More time

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.**

- When we have pure data in the node. That is, all the data in the node belongs to the same class.

- When the tree reaches the maximum depth. To prevent overfitting and reduce the complexity, we will limit the tree's depth, so we stop splitting when we get the limitation.

- When the split doesn't improve the information gain, the child node will be the same as the parent node. This split cannot let the node receive more information gain and make the model perform better. It only increases the tree size and complexity. Hence, we should stop splitting.

- When the data in the node reaches the minimum number. To ensure the node can give a reliable prediction, there must has enough data in the node. Thus, we will limit the minimum data in the node, and stop splitting if the number of data is less than a specific number.

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.**

No, I don’t agree with the student. The key to making random forests perform well is the diversity of the trees. Every tree has a low correlation with each other, so in the final prediction, every tree can provide a different suggestion. Combining all the predictions, we can have a good final prediction.

If we set m = 1, every tree will only consider one feature instead of a random set of features. This will decrease the diversity in the trees and every tree is similar, since they all make the prediction based on one feature. The correlation between the trees reduces the effect of randomness, thus reducing the accuracy.

In addition, if the trees only consider one feature to make the decision, it will be too simple and likely to underfit. To elaborate, the model cannot capture the full complexity of the data, so it will not be able to fully understand the relationships between features. As a result, it cannot have a good performance.

Also, the student mentions that setting m = 1 will reduce variance and improve accuracy. Although it can somehow reduce the complexity of the tree and therefore reduce the variance, it actually increases the bias, which lowers the overall accuracy.

In conclusion, the statement that setting m to 1 will improve the accuracy while reducing the variance is wrong. It will make the model underfit and give a bad prediction. So it is important to decide a proper number of features used to split the tree, which allows the tree to be diverse and capture enough information to make the prediction.