NYCU Introduction to Machine Learning, Homework 3

111550113, 謝詠晴

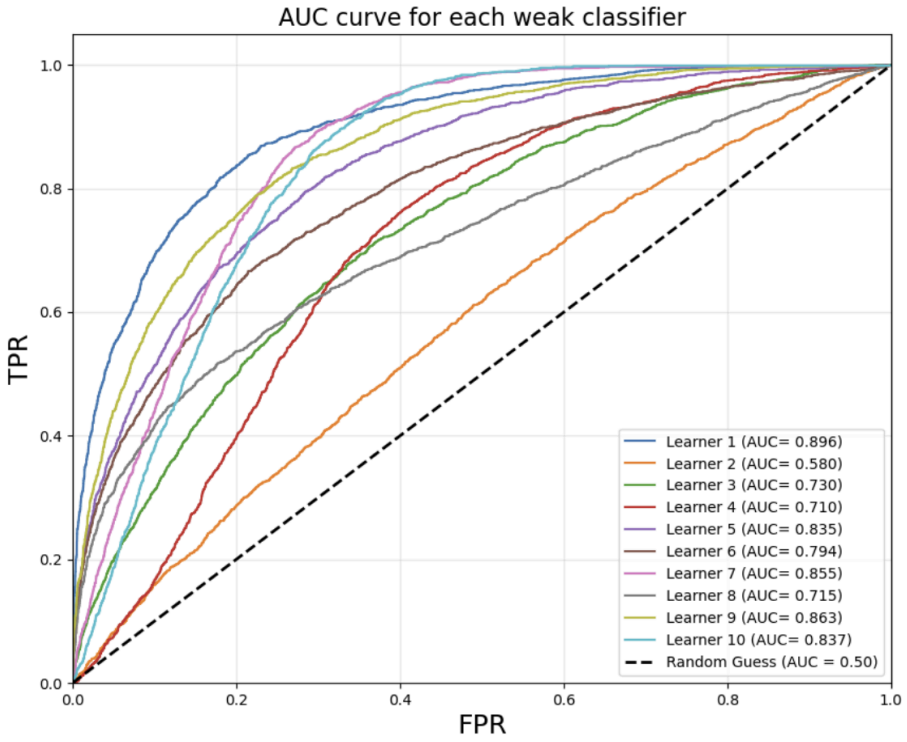
Part. 1, Coding (60%):

(20%) Adaboost

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

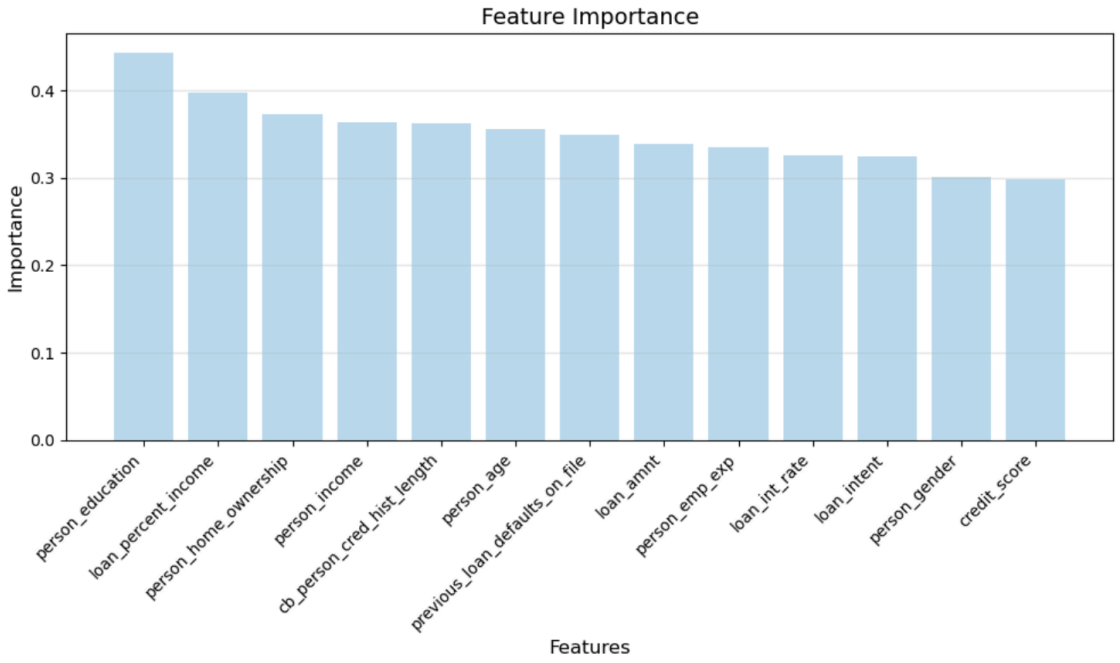
```
2024-11-18 14:33:47.265 | INFO     | __main__:main:54 - AdaBoost - Accuracy: 0.8590
```

2. (5%) Plot the AUC curves of each 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"""
        num_features = self.learners[0].model[0].weight.shape[1]
        feature_importance = torch.zeros(num_features)
        for alpha, model in zip(self.alphas, self.learners):
            with torch.no_grad():
                weights = model.model[0].weight.abs()
                weights_2 = model.model[1].weight.abs()
                # neuron_importance = weights.sum(dim=0)
                # print(f"weights shape: {weights.shape}, weights_2 shape: {weights_2.shape}")
                combined_importance = torch.matmul(weights_2, weights)
                neuron_importance = combined_importance.squeeze(0)
                feature_importance += alpha * neuron_importance
        return feature_importance.tolist()
```

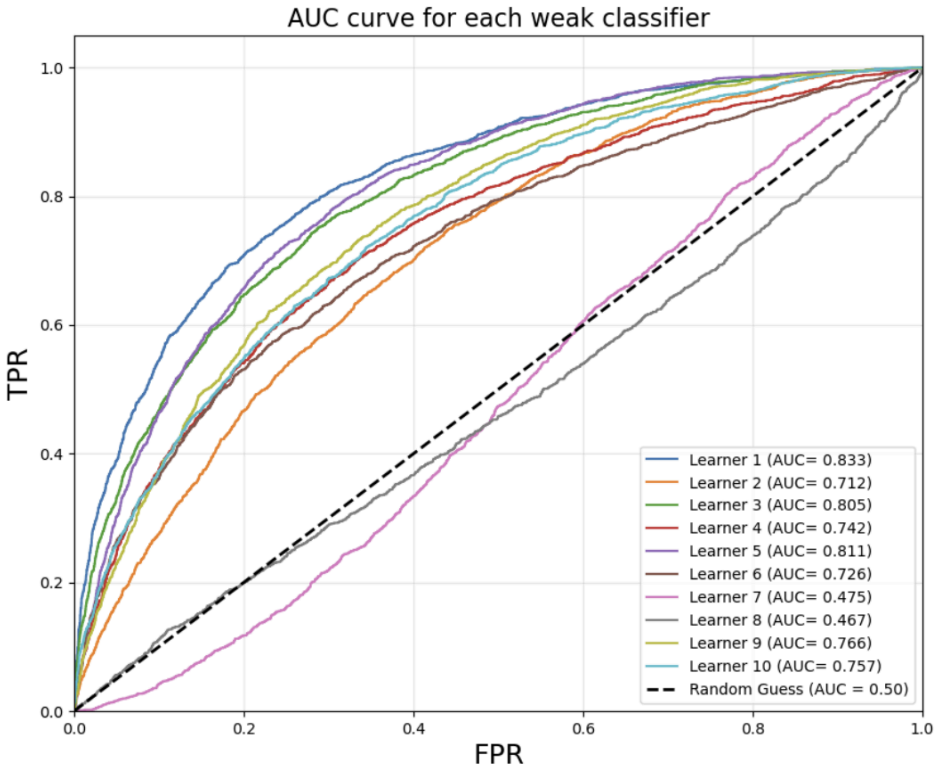


(20%) Bagging

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

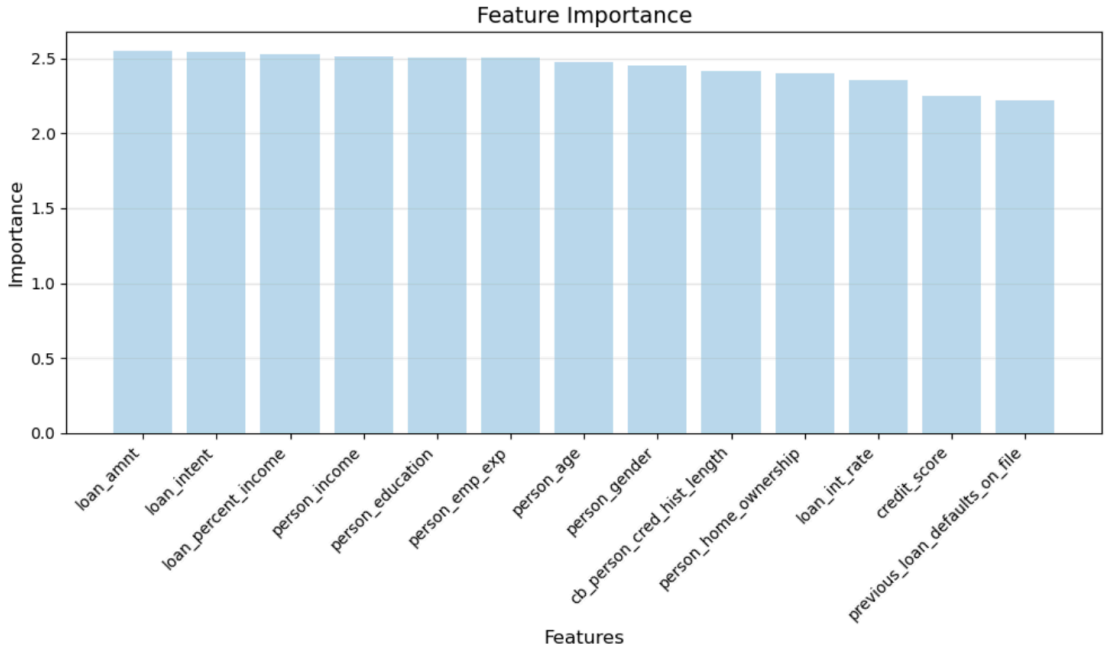
```
2024-11-18 13:35:11.934 | INFO     | __main__:main:76 - Bagging - Accuracy: 0.8441
```

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"""
    num_features = self.learners[0].model[0].weight.shape[1]
    feature_importance = torch.zeros(num_features)
    for model in self.learners:
        with torch.no_grad():
            weights = model.model[0].weight.abs()
            weights_2 = model.model[1].weight.abs()
            combined_importance = torch.matmul(weights_2, weights)
            neuron_importance = combined_importance.squeeze(0)
            # neuron_importance = weights.sum(dim=0)
            feature_importance += neuron_importance
    return feature_importance.tolist()
```



(15%) Decision Tree

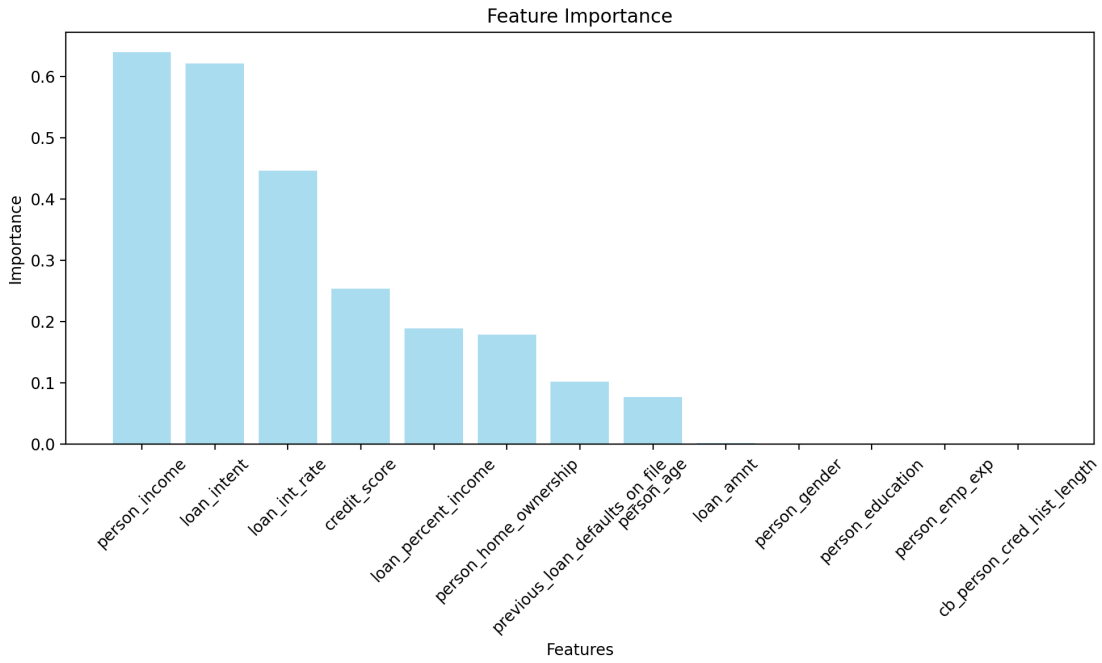
7. (5%) Compute the Gini index and the entropy of the array [0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1].

```
(ml) PS C:\Users\yungching\Desktop\ML\HW3\release-113_1> python ./main.py
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]: 0.9456603045232519
```

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

```
2024-11-18 13:36:42.316 | INFO     | __main__:main:96 - DecisionTree - Accuracy: 0.9040
```

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).

```
(ml) PS C:\Users\yungching\Desktop\ML\HW3\release-113_1> flake8 main.py
(ml) PS C:\Users\yungching\Desktop\ML\HW3\release-113_1>
```

```
(ml) PS C:\Users\yungching\Desktop\ML\HW3\release-113_1> flake8 src/
(ml) PS C:\Users\yungching\Desktop\ML\HW3\release-113_1>
```

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.

   1) Bagging:
   Randomly choosing multiple subsets of the training data with replacement and training a weak learner on each subset. Finally using majority vote or average to aggregate the prediction.
   2) Boosting:
   Boosting involves training weak learners sequentially, with higher weights assigned to misclassified instances in each iteration. Therefore, the subsequent learners can focus more on the instances that are difficult to classify. At the end, each learner is assigned a weight(alpha) to calculate the final prediction.
   3) Differences:

	Bagging	Boosting
Training	Parallel	Sequential
Data	Random sampling with replacement	Weighted sampling

Model focus	Reducing variance	Reducing bias
Weak Learner Importance	Equal weight	Weighted based on performance

4) Comparison:
   - Bagging is typically more robust and less prone to overfitting while boosting may achieve higher accuracy but requires more careful tuning.
   - Because bagging can be implemented parallel, it is more efficient for larger datasets. Boosting needs sequential training, which can be computationally expensive.
   - Boosting can be more sensitive to noise and outliers.

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) Maximum tree depth:
      A deeper tree can capture more complex patterns, but if it becomes too deep, it is likely to overfit. Therefore, it is important to limit the depth to prevent this from happening.
   2) Minimum number of data per leaf:
      By limiting the number of samples per leaf, we can make sure that leaf nodes have sufficient data points to make reliable predictions and avoid overfitting.
   3) Information gain is less than a threshold value:
      Information gain below a threshold: If the information gain of a split is below the threshold, it is considered insufficient to improve the model significantly

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.
   I do not agree with the student's claim. While setting m = 1 may reduce the correlation between individual trees, making them more diverse, trees that only use one feature can become overly simplistic and may fail to capture complex relationships in the data, leading to poor predictions. This could negatively impact overall performance. For example, if a dataset has many features but only a few are strongly correlated with the target variable, choosing m = 1 may not be the ideal approach. Overall, the optimization of random forests involves finding a balance between diversity and model capacity.