NYCU Introduction to Machine Learning, Homework 3

110550080, 何曉嫻

Part. 1, Coding (50%):

(30%) 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].

```
Part 1: Decision Tree
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]: 0.9456603046006401
```

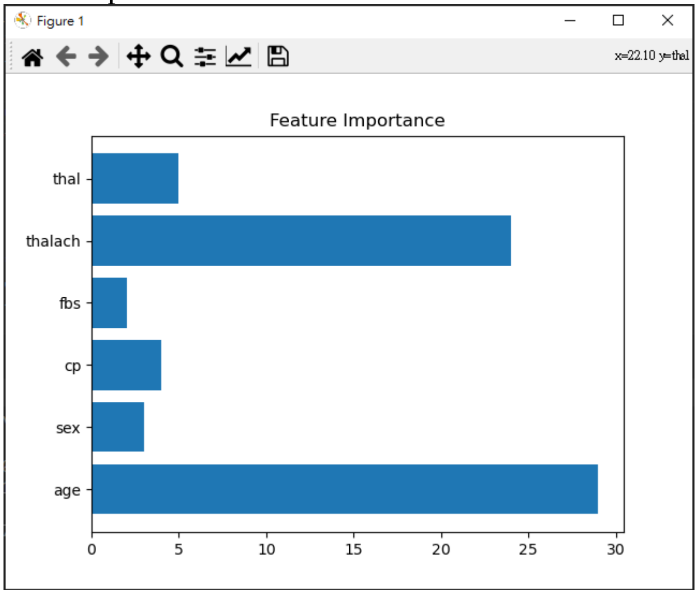
2. (10%) Show the accuracy score of the testing data using criterion="gini" and max_depth=7. Your accuracy score should be higher than 0.7.

```
Accuracy (gini with max_depth=7): 0.7049180327868853
```

3. (10%) Show the accuracy score of the testing data using criterion="entropy" and max_depth=7. Your accuracy score should be higher than 0.7.

```
Accuracy (entropy with max_depth=7): 0.7213114754098361
```

4. Train your model using criterion="gini", max_depth=15. Plot the feature importance of your decision tree model by simply counting the number of times each feature is used to split the data.



(20%) Adaboost

5. (20%) Tune the arguments of AdaBoost to achieve higher accuracy than your Decision Trees.

```
Part 2: AdaBoost
Accuracy: 0.8032786885245902
```

Part. 2, Questions (50%):

1. (10%) True or False. If your answer is false, please explain.
   a. (5%) In an iteration of AdaBoost, the weights of misclassified examples are increased by adding the same additive factor to emphasize their importance in subsequent iterations.
      False.
      The weights of misclassified examples are not adjusted by adding the same additive factor uniformly. Instead, it is typically determined based on the algorithm's logic, often involving a weight adjustment proportional to the misclassification error.
   b. (5%) AdaBoost can use various classification methods as its weak classifiers, such as linear classifiers, decision trees, etc.
      True.

2. (10%) How does the number of weak classifiers in AdaBoost influence the model's performance? Please discuss the potential impact on overfitting, underfitting, computational cost, memory for saving the model, and other relevant factors when the number of weak classifiers is too small or too large.

   (1) Too few weak classifiers might result in underfitting, where the model fails to capture the complexities in the data. It may have poor performance due to a lack of modeling power. Too many weak classifiers can lead to overfitting. The model might memorize the training data, capturing noise and outliers, and therefore might perform poorly on unseen data.
   (2) Having many weak classifiers increases computational cost, as each iteration involves training multiple weak learners sequentially. This can become computationally expensive, especially when using complex weak learners or dealing with large datasets.
   (3) Storing a large ensemble of weak classifiers consumes more memory. The memory required to store the model increases with the number of weak learners.
   (4) A larger number of weak classifiers may reduce the interpretability of the model. Interpretability decreases as the complexity of the ensemble increases, making it hard to understand the decision-making process of the model.

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, it is not guaranteed to improve accuracy or reduce variance. Setting m = 1 can increase the diversity among the trees in the forest. However, restricting the choice of features may result in trees that are not as informative or effective in capturing the

true relationships present in the data. Consequently, this could lead to a decrease in overall accuracy.
Besides, it may increase the variance. Each decision tree built on a single feature at each node might become more sensitive to noise or spurious correlations in that specific feature. Leading to higher variance among the predictions of individual trees, and overfitting.

4. (15%) The formula on the left is the forward process of a standard neural network while the formula on the right is the forward process of a modified model with a specific technique.

$$z^{(l+1)} = w^{(l+1)} y^l + b^{(l+1)}$$

$$y^{(l+1)} = f(z^{(l+1)})$$

$$r^l = Bernoulli(p)$$

$$\tilde{y}^l = r^l y^l$$

$$z^{(l+1)} = w^{(l+1)} \tilde{y}^l + b^{(l+1)}$$

$$y^{(l+1)} = f(z^{(l+1)})$$

a. (5%) According to the two formulas, describe what is the main difference between the two models and what is the technique applied to the model on the right side.
The main difference is the model on the right will randomly drop units from the neural network during training. For any layer l, r^l is a vector of independent Bernoulli random variables each of which has probability p of being 1. This vector is sampled and multiplied element-wise with the outputs of that layer y^l, to create the thinned outputs $\tilde{y}^l$. The thinned outputs are then used as input to the next layer.

b. (10%) This technique was used to deal with overfitting and has many different explanations; according to what you learned from the lecture, try to explain it with respect to the ensemble method.
Dropout can be understood in relation to ensemble methods, specifically the concept of creating diverse models to reduce overfitting. Ensemble methods, like boosting, aim to reduce overfitting by training multiple models and combining their predictions.
This technique introduces an ensemble of thinned networks within a single model during training. It forces the network to learn redundant representations by preventing units from co-adapting, thus creating multiple "thinned" versions of the network.
Each thinned network can be considered as a different sub-model that captures different aspects of the data due to the dropped units, like an ensemble of models trained on different subsets of the data. At test time, the predictions are made by averaging the outputs of all these thinned networks, effectively creating an ensemble of models within a single network.