NYCU Introduction to Machine Learning, Homework 2

110652021，龔大承

The screenshot and the figures we provided below are just examples. **The results below are not guaranteed to be correct.** Please make sure your answers are clear and readable, or no points will be given. Please also remember to convert it to a pdf file before submission. You should use English to answer the questions. After reading this paragraph, you can delete this paragraph.

Part. 1, Coding (50%):

For this coding assignment, you are required to implement the Decision Tree and Adaboost algorithms using only NumPy. After that, train your model on the provided dataset and evaluate the performance on the testing data.

(30%) Decision Tree

Requirements:

- Implement **gini index** and **entropy** for measuring the best split of the data.
- Implement the decision tree classifier (CART, Classification and Regression Trees) with the following two arguments:
- **criterion**: The function to measure the quality of a split of the data. Your model should support "gini" and "entropy".
- **max_depth**: The maximum depth of the tree. If max_depth=None, then nodes are expanded until all leaves are pure. max_depth=1 equals to splitting data once.

Tips:

- Your model should produce the same results when rebuilt with the same arguments, and there is no need to prune the trees.
- You can use the recursive method to build the nodes.

Criteria:

1. (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]: 0.9456603046006401
```

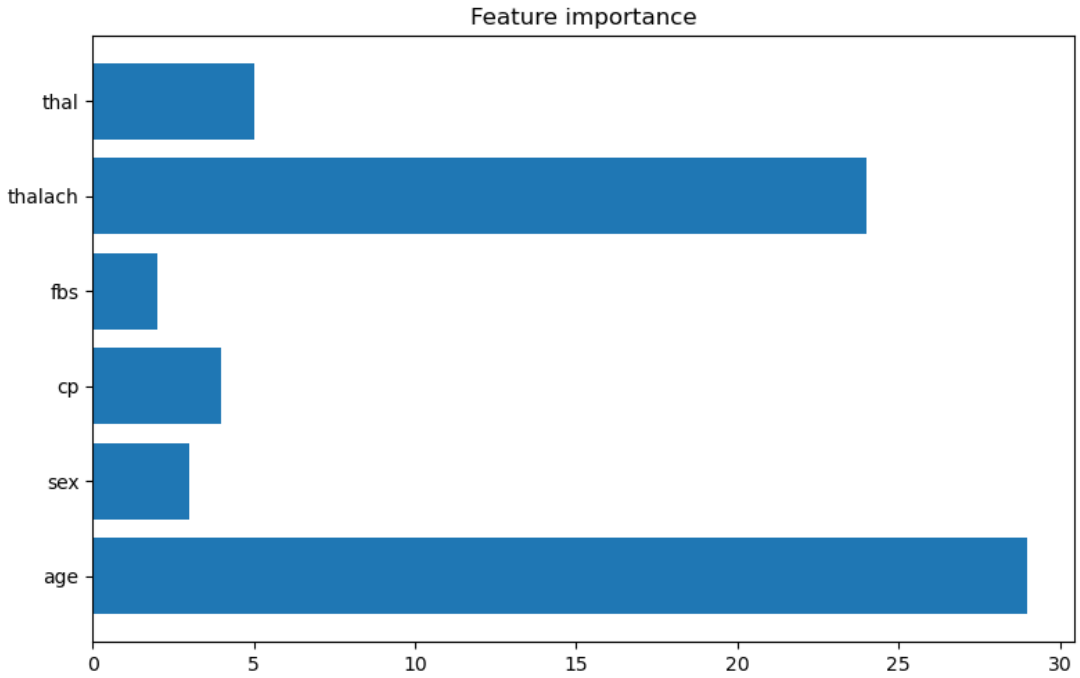
2. (10%) Show the accuracy score of the testing data using criterion="gini" and max_depth=7. Your accuracy score should be higher than 0.7.

```
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. (5%) 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. Your answer should look like the plot below:



(20%) Adaboost

Requirements:

- Implement the Adaboost algorithm by using the decision tree classifier (max_depth=1) you just implemented as the weak classifier.
- The Adaboost model should include the following two arguments:
- **criterion**: The function to measure the quality of a split of the data. Your model should support "gini" and "entropy".
- **n_estimators**: The total number of weak classifiers.

Tips:

- You can set any random seed to make your result reproducible.

Criteria:

1. (20%) Tune the arguments of AdaBoost to achieve higher accuracy than your Decision Trees.

```
Part 2: AdaBoost
Accuracy: 0.819672131147541
```

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.

       True, but with a clarification. In an iteration of AdaBoost, the weights of misclassified examples are indeed increased, but not by adding the same additive factor. Instead, they are updated by multiplying them with a factor that depends on the error rate of the weak classifier. This factor is an exponential function of the error rate, which means that the weights of misclassified examples increase exponentially as the error rate increases. This ensures that these examples are given more importance in subsequent iterations.

    b. (5%) AdaBoost can use various classification methods as its weak classifiers, such as linear classifiers, decision trees, etc.

       True. AdaBoost is a meta-algorithm, and as such, it can use any machine learning algorithm as a weak learner or base classifier. This includes, but is not limited to, linear classifiers, decision trees, and even neural networks. The only requirement is that the base classifier must be better than random guessing. Decision trees, especially ones with a small depth, are commonly used because they are simple, fast to train, and often provide a good trade-off between bias and variance.

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.

   The number of weak classifiers in AdaBoost can significantly influence the model's performance:

   Underfitting: If the number of weak classifiers is too small, the AdaBoost model may not capture the complexity of the data well, leading to underfitting. The model might have high bias and low variance, resulting in poor performance on both the training and test sets.

Overfitting: On the other hand, if the number of weak classifiers is too large, the model may become overly complex and start to fit the noise in the training data, leading to overfitting. However, AdaBoost is less prone to overfitting compared to some other algorithms, as it focuses more on the hard-to-classify instances.

Computational Cost: The computational cost of training and prediction increases with the number of weak classifiers. Each additional classifier requires its own training and prediction time, so a model with many weak classifiers can be slow to train and use.

Memory Usage: The memory required to store the model also increases with the number of weak classifiers. Each classifier has its own parameters that need to be stored, so a model with many weak classifiers can require a lot of memory.

Model Performance: Initially, adding more weak classifiers tends to improve the model's performance, as it allows the model to capture more complex patterns in the data. However, after a certain point, adding more weak classifiers may not improve the performance significantly and might even degrade it due to overfitting.

In conclusion, choosing the right number of weak classifiers is a trade-off between underfitting and overfitting, computational cost, and memory usage. It's often determined using cross-validation or other model selection techniques.

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.

The student's idea of setting m (the number of random features used in each node of each decision tree) to 1 in a Random Forest model is interesting, but it may not necessarily lead to improved accuracy and reduced variance as claimed.

Random Forests work by creating a large number of decision trees, each trained on a different subset of the data and using a different subset of the features at each node. The idea is to create diverse trees that make different kinds of errors,

which can then be averaged out when the trees "vote" on the final prediction, reducing the overall variance.

Setting m to 1 would indeed increase the diversity of the trees, as each node would be making a decision based on a single feature. However, this could also significantly reduce the power of each individual tree. Each tree would be making decisions based on very limited information, which could lead to high bias and poor performance.

Furthermore, one of the strengths of Random Forests is their ability to capture interactions between features. If each node is only considering one feature, this ability is lost.

In terms of variance, while it's true that averaging a large number of diverse trees can reduce variance, this is only beneficial if the trees are accurate to begin with. If the trees are too weak (as they might be if m is set to 1), the model's performance could suffer.

In conclusion, while the student's idea is interesting and might work well in certain situations, it's not a guaranteed way to improve accuracy and reduce variance in a Random Forest model. As with many things in machine learning, the best approach would likely depend on the specific dataset and problem at hand.

(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.

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 lies in the introduction of a stochastic element, $r(l)$, and the multiplication of $r(l)$ with the activations $y(l)$. This results in a stochastically masked version $y(\sim l)$ that is used in the computation of the next layer's inputs.

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.

The technique applied to the model on the right is dropout regularization. Dropout involves randomly setting a fraction of input units to zero at each update

during training. The Bernoulli random variable $r(l)$ with probability p determines which neurons are "dropped out" (i.e., their activations are set to zero) in each forward pass.

Dropout can be seen as a form of ensemble learning. Ensemble methods, like bagging and boosting, involve combining the predictions of multiple models to improve overall performance. Dropout, by randomly dropping out neurons during training, can be seen as training different subnetworks within the overall network. During testing, all these subnetworks contribute to the final prediction, mimicking the idea of an ensemble of models. This helps prevent overfitting by introducing diversity in the trained models and improving generalization.

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

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

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

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

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

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