NYCU Introduction to Machine Learning, Homework 3

109550198 卜銳凱

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.

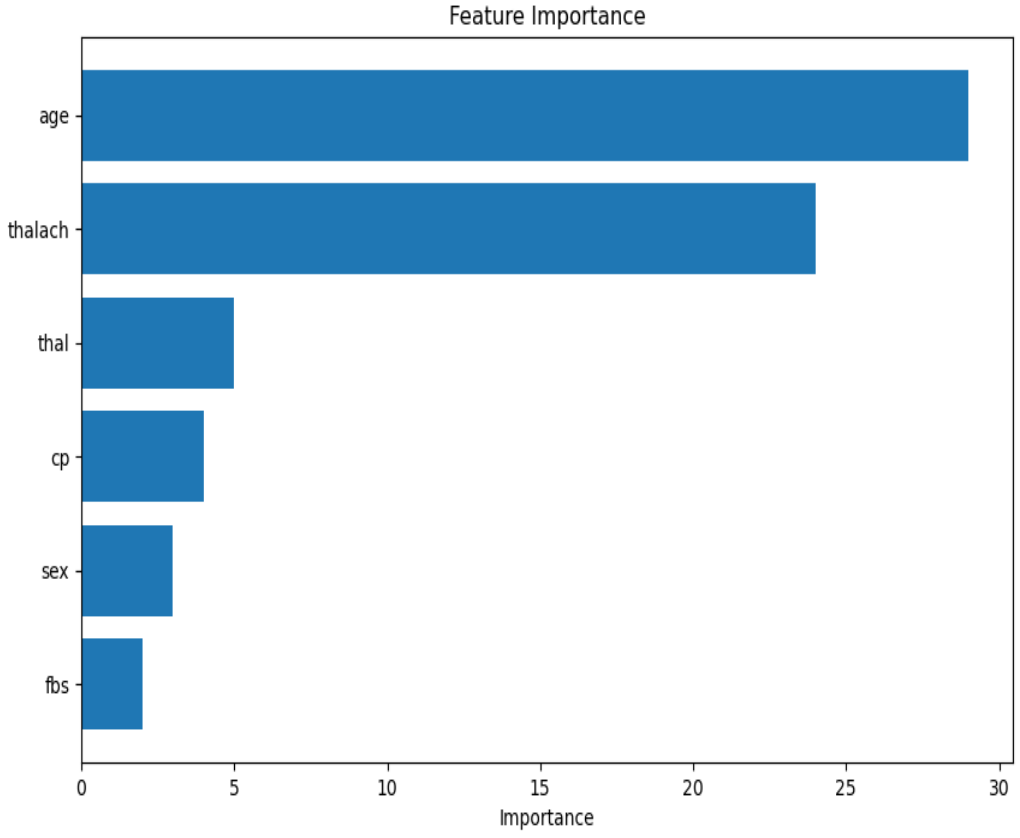
```
Accuracy (gini with max_depth=7): 0.7213114754098361
```

(next page)

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:



(next page)

(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:

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

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

Points	Testing Accuracy
20 points	0.8 <= acc
15 points	0.78 <= acc < 0.8
10 points	0.76 <= acc < 0.78
5 points	**0.74 <= acc < 0.76**
0 points	**acc < 0.74**

(next page)

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 simply increased by adding the same additive factor in an AdaBoost iteration. Instead, the weights are updated by multiplying the weights of misclassified examples by a factor $\exp(\alpha)$, where α is a value calculated during the boosting process that is related to the error rate of the weak learner. This process increases the weights of the misclassified examples, thus emphasizing their 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 an ensemble method that can use several types of classification algorithms as its weak classifiers. These weak classifiers can be linear classifiers, decision trees, or any other method that receives weights on training instances and provides a model for which predictions can be made. The selection of weak classifiers is determined on the implementation and problem domain.

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 is a crucial parameter that influences the model's performance in several ways:

   Overfitting: If there are too many weak classifiers, AdaBoost may overfit the training data. This means that, while it will perform well on training data, it may not generalize well to untested data. Overfitting occurs when the model becomes too complex and begins capturing noise in the training data as if it were a valid pattern.

   Underfitting: The model may underfit if there are too few weak classifiers. Underfitting occurs when a model is too simple and fails to capture the underlying patterns in the data, leading to poor performance on both training and unseen data.

(next page)

Computational Cost: The higher the computational cost, the higher the number of weak classifiers is. AdaBoost iterations involve adjusting weights, selecting a weak classifier, and updating its parameters, all of which can be computationally intensive. This has an impact on training time as well because more classifiers mean more iterations.

Memory for Saving the Model: A larger number of weak classifiers increases the size of the final model, requiring more memory for storage. This can be a concern when deploying models in environments with limited memory resources.

Model Complexity and Interpretability: As the number of weak classifiers increases, the model becomes more complex and harder to interpret. AdaBoost combines the decisions of all the weak classifiers, making it difficult to understand each classifier's contribution to the final a decision.

Error Reduction and Margins: Initially, when more classifiers are added, the error on the training set typically decreases, and the classification margins (how confidentially each point is classified) improve. However, beyond a certain point, adding more classifiers may not result in significant error reduction and can even start hurting margins by focusing too much on the most challenging cases.

Diminishing Returns: There is frequently a point of diminishing returns where adding additional classifiers does not significantly increase the model's capacity to generalize to new data. Beyond this point, the advantages of adding more classifiers are outweighed by the costs and risks of overfitting.

The key is to find a balance where the number of weak classifiers is sufficient to capture the complexity of the data without overfitting. This can be done through cross-validation, where the performance of the AdaBoost model is validated on a separate set of data not used during training, and by tracking performance metrics as more classifiers are added. Techniques like early stopping can also be employed, where training is halted once the improvement of the model on the validation set becomes negligible.

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.

(next page)

The student's idea of setting $m=1$ in a random forest, where m is the number of features considered for splitting at each node, is not likely to make random forests more powerful for several reasons:

Diversity vs. Relevance: While it is true that random forests rely on tree diversity, this diversity must be balanced with relevance. By setting m=1, each split in each tree would only consider a single, random feature. This severely limits the trees' ability to find the most informative splits because the chance of selecting the most relevant feature at each split is very low, especially if the dataset contains many features.

Strength of Individual Trees: Individual decision trees in a random forest should still be strong learners, albeit less connected with one another. By only considering one feature at each split, you significantly weaken the individual trees, making them barely better than a random guess, especially in complex datasets which included features that individually have little predictive power.

Reduction in Variance: Random forests reduce variance by aggregating less correlated trees, rather than increasing bias. Setting m=1 will likely increase the bias of individual trees, which might not be offset by the variance reduction from averaging many such high-bias trees.

Accuracy: A random forest's accuracy depends by the strength of the individual trees as well as the correlation between them. If the trees are too weak (as they would be with m=1), the forest will be less accurate because the errors of the weak learners' compound rather than cancel out.

Dimensionality and Feature Interactions: In high-dimensional spaces or where feature interactions are important, considering only one feature at a time ignores these interactions and the joint distribution of the features, which could be critical for making accurate predictions.

Empirical Evidence: The strength of random forests comes in their ability to accommodate many weakly correlated, reasonably strong decision trees. There is substantial empirical evidence that setting m to a value such as the square root of the number of features or a small constant fraction of the number of features often results in good performance. Setting m=1 is a radical departure from this practice and is not supported by empirical evidence.

For these reasons, the student's claim does not hold up well against the theory and practice behind random forests. It is generally a better idea to set m to a value that allows for some diversity in the trees while still ensuring that the individual trees

(next page)

are strong and that the splits are informative.

(next page)

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.
    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 between the two formulas on the left and right side seems to be the inclusion of a Bernoulli random variable r' in the formula on the right side. This r' is multiplied element-wise by the output y of the previous layer to give y', which then feeds into the next layer's computations. The technique applied to the model on the right is known as Dropout. Dropout randomly sets a proportion of the elements in y to zero during training, which is controlled by the parameter p in the Bernoulli distribution. The purpose of this is to prevent the network from becoming overly dependent on any single neuron, thus mitigating overfitting.

    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 as a technique is indeed used to combat overfitting, which is when a model learns the training data too well, including the noise and outliers, and performs poorly on new, unseen data. Overfitting can lead to models that have high accuracy on training data but fail to generalize to test data. By dropping out random neurons during training, it forces the network to learn more robust features that are not reliant on any small set of neurons. This can be thought of as a form of ensemble learning because at each training stage, a different "thinned" network is created, which can be seen as a unique member in an ensemble of networks. At test time, all neurons are used, but their outputs are scaled by p to account for the reduced number of units during training. This is akin to averaging the predictions of all possible thinned networks.

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

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

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

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

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

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