NYCU Introduction to Machine Learning, Homework 3

109350008, 張詠哲

Deadline: Nov. 28, 23:59

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
```

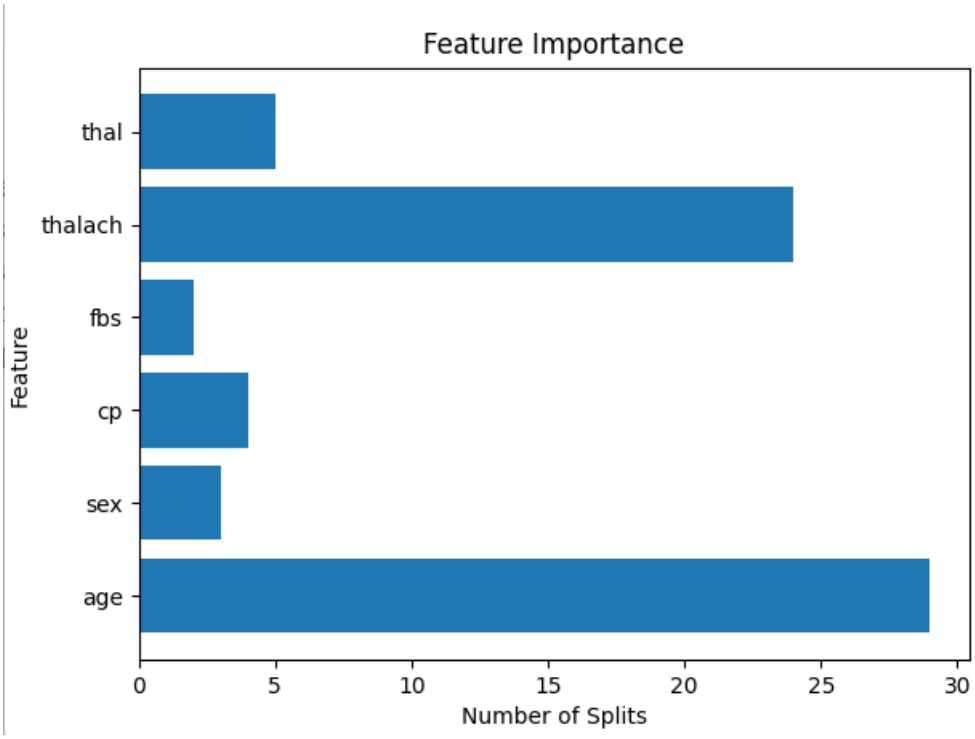
2. (10%) Show the accuracy score of the testing data using criterion="gini" and max_depth=7. Your accuracy score should be higher than 0.7.

```
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.



(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 sup port "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 Tre es.

```
ada = AdaBoost(criterion = 'entropy', n_estimators = 10)
```

```
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

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.

      Ans. False

      In AdaBoost, the weights of misclassified examples are increased, but the increase is not the same additive factor for all misclassified examples. The weights are adjusted such that the misclassified examples receive more emphasis in the subsequent iterations. If an example is misclassified, its weight is increased, and if it is correctly classified, its weight is decreased.

      The formula is weights *= np.exp(-alpha * y * y_pred) / np.sum(weights)

      (alpha is the weight of the weak classifier, y is the true label, and y_pred is the predicted label)

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

      Ans. 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.
   - Overfitting and underfitting:

- Too few weak classifiers (underfitting):
  If the number of weak classifiers is too small, the model may underfit the data, meaning it might hard to capture the underlying patterns in the training data. ha ving too few of them may limit the model's ability to represent the complexity o f the data.
- Too many weak classifiers (overfitting):
  If the number of weak classifiers is too large, the model may start memorizing t he training data, leading to overfitting. the model could become too specific to t he training set, performing poorly on unseen data.

- Computational cost:
  - Too few weak classifiers:
    Training a model with too few weak classifiers is computationally less expensiv e since performing fewer iterations and updating weights fewer times.

  - Too many weak classifiers:
    As the number of weak classifiers increases, the computational cost of training t he model also increases. each additional weak classifier requires training, and th e overall training time may become a limiting factor (especially for complex we ak classifiers).

- Memory usage:
  - Too few weak classifiers:
    The memory required to store a model with few weak classifiers is lower. the m odel is simpler and requires less information to represent.
  - Too many weak classifiers:
    A model with a large number of weak classifiers may consume more memory, a s it need to store information about each weak classifier and its associated weigh ts.

- Algorithm stability:
  - Too few weak classifiers:
    Adaboost may struggle to converge or may converge to a suboptimal solution w hen the number of weak classifiers is insufficient.
  - Too many weak classifiers:

The algorithm may become more sensitive to noise in the data, leading to instability. it might start fitting the noise in the training data rather than capturing the true underlying patterns.

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.

In my opinion. The student's proposal (set m=1 in a random forest, where m is the number of random features used in each node of each decision tree) is not likely to improve accuracy or reduce variance. On the contrary, it may have adverse effects on the performance of the random forest. There are my reason:

- Correlation and diversity:
  Random forests achieve high performance by building diverse trees, where each tree is trained on a random subset of features. Setting m=1 means that each decision tree in the forest only considers a single random feature at each split. This lack of diversity can lead to trees that are highly correlated, as they are all making decisions based on the same feature at each split. Random forests benefit from the diverse trees, and overly correlated trees may not provide significant ensemble improvements.

- Feature importance and robustness:
  Random forests rely on the idea of feature importance, where different features contribute differently to the predictive power of the model. Constraining each tree to use only one random feature at each node limits the model's ability to capture the importance of multiple features. This can lead to less robust and less accurate predictions.

- Variance reduction:
  Random forests reduce variance by averaging the predictions of multiple trees. The diversity of the trees allows them to capture different aspects of the data and reduce the overall variance. Setting m=1 limits the diversity, potentially increasing the overall variance (especially if the selected feature is not highly informative for the given task).

- Performance trade-off:

The trade-off between bias and variance is a crucial role in machine learning. While r educing the number of features used in each node may reduce variance, it also increas es bias.

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

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.

Ans.

The main difference between 2 models is that on the right side formula, the y^l times a binary mask r^l, which is generated from Bernoulli distribution with pro bability p. The element-wise multiplication of r^l and y^l ($\tilde{y}^l = r^l \cdot y^l$) to introduce s a form of regularization. And the technique applied is dropout. Where randomly selected neurons are "dropped out" during training. This helps prevent coadaptatio n of hidden units and reduces overfitting.

b. (10%) This technique was used to deal with overfitting and has many different exp lanations; according to what you learned from the lecture, try to explain it with res pect to the ensemble method.

Ans.

Dropout is a regularization technique that involves randomly setting a fract ion of input units to zero during each update of the model. This helps prevent comp lex co-adaptations on training data and reduces overfitting.

The ensemble perspective arises from the idea that, during training, the net work is exposed to different subsets of neurons being "active" or "dropped out." Ea ch subset corresponds to a different configuration of the network, and the final net work can be viewed as an ensemble of these different configurations. At test time, dropout is typically turned off, and the predictions are made by considering the ave rage behavior of all these configurations. Dropout helps create a more robust mode l by preventing reliance on specific neurons and encourages the network to learn m ore general features. It can be interpreted as a form of model averaging during train ing, which is a key principle of ensemble methods.

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

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

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

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

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

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