NYCU Pattern Recognition, Homework 3

411551005 徐浩哲

Part. 1 Coding (80%)

(30%) Decision Tree

1. (5%) Compute the Entropy and Gini index of the array provided in the sample code, using the formulas on page 6 of the HW3 slide.

```
[4]: ex1 = np.array(["+", "+", "+", "+", "+", "-"])
     ex2 = np.array(["+", "+", "+", "-", "-", "-"])
     ex3 = np.array(["+" ,"-", "-", "-", "-", "-"])
     print(f"{ex1}: entropy = {entropy(ex1)}\n{ex2}: entropy = {entropy(ex2)}\n{ex3}: entropy = {entropy(ex3)}\n")
     print(f"{ex1}: gini index = {gini(ex1)}\n{ex2}: gini index = {gini(ex2)}\n{ex3}: gini index = {gini(ex3)}\n")
```

```
['+' '+' '+' '+' '+' '-']: entropy = 0.6500224216483541
['+' '+' '+' '-' '-' '-']: entropy = 1.0
['+' '-' '-' '-' '-' '-']: entropy = 0.6500224216483541

['+' '+' '+' '+' '+' '-']: gini index = 0.2777777777777777
['+' '+' '+' '-' '-' '-']: gini index = 0.5
['+' '-' '-' '-' '-' '-']: gini index = 0.2777777777777777
```

2. (10%) Show the accuracy score of the validation data using criterion='gini' and max_features=None for max_depth=3 and max_depth=10, respectively.
3. (10%) Show the accuracy score of the validation data using max_depth=3 and max_features=None, for criterion='gini' and criterion='entropy', respectively.

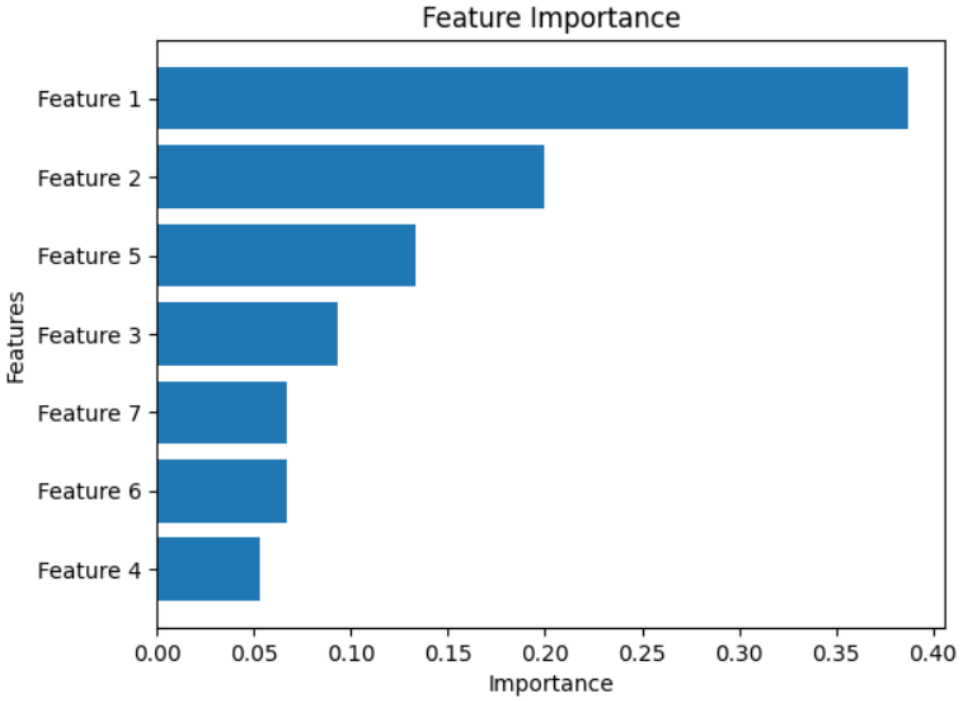
```
[7]: Tree = MyDecisionTreeClassifier(criterion= "gini", max_depth=3, n_feats= None)
     Tree.fit(X_train, y_train)
     acc = accuracy_score(y_val, Tree.predict(X_val))
     print(f'Q2-1 max_depth= 3: {acc:.2f}')

     Tree = MyDecisionTreeClassifier(criterion= "gini", max_depth=10, n_feats= None)
     Tree.fit(X_train, y_train)
     acc = accuracy_score(y_val, Tree.predict(X_val))
     print(f'Q2-2 max_depth= 10: {acc:.2f}')

     Tree = MyDecisionTreeClassifier(criterion= "gini", max_depth=3, n_feats= None)
     Tree.fit(X_train, y_train)
     acc = accuracy_score(y_val, Tree.predict(X_val))
     print(f'Q3-1 criterion= gini: {acc:.2f}')

     Tree = MyDecisionTreeClassifier(criterion= "entropy", max_depth=3, n_feats= None)
     Tree.fit(X_train, y_train)
     acc = accuracy_score(y_val, Tree.predict(X_val))
     print(f'Q3-2 criterion= entropy: {acc:.2f}')

     # For Q2-1, validation accuracy should be higher than or equal to 0.73
     # For Q2-2, validation accuracy should be higher than or equal to 0.85
     # For Q3-1, validation accuracy should be higher than or equal to 0.73
     # For Q3-2, validation accuracy should be higher than or equal to 0.77
```

```
Q2-1 max_depth= 3: 0.73
Q2-2 max_depth= 10: 0.86
Q3-1 criterion= gini: 0.73
Q3-2 criterion= entropy: 0.77
```

4. (5%) Train your model using criterion='gini', max_depth=10 and

max_features=None. Plot the feature importance of your decision tree model by simply counting the number of times each feature is used to split the data.

```
[8]: np.random.seed(0)
     Tree = MyDecisionTreeClassifier(criterion= "gini", max_depth=10, n_feats=None)
     Tree.fit(X_train, y_train)
     acc = accuracy_score(y_val, Tree.predict(X_val))
     Tree.plot_feature_importance()
```



5. (10%) Show the accuracy score of the validation data using criterion='gini', max_depth=None, max_features=sqrt(n_features), and bootstrap=True, for n_estimators=10 and n_estimators=50, respectively.

```
[9]: np.random.seed(1)
     rf_estimators10 = RandomForestClassifier(n_estimators=10, max_features=int(np.sqrt(X_train.shape[1])), bootstrap=True
                                              , criterion='gini', max_depth=None)
     rf_estimators10.fit(X_train, y_train)
     acc = accuracy_score(y_val, rf_estimators10.predict(X_val))
     print(f'Q5-1 n_estimators=10: {acc:.2f}')
     # For Q5-1, validation accuracy should be higher than or equal to 0.88
```

```
Q5-1 n_estimators=10: 0.88
```

```
[10]: np.random.seed(1)
      rf_estimators10 = RandomForestClassifier(n_estimators=50, max_features=int(np.sqrt(X_train.shape[1])), bootstrap=True
                                               , criterion='gini', max_depth=None)
      rf_estimators10.fit(X_train, y_train)
      acc = accuracy_score(y_val, rf_estimators10.predict(X_val))
      print(f'Q5-2 n_estimators=50: {acc:.2f}')
      # For Q5-2, validation accuracy should be higher than or equal to 0.89
```

```
Q5-2 n_estimators=50: 0.89
```

6. (10%) Show the accuracy score of the validation data using criterion='gini', max_depth=None, n_estimators=10, and bootstrap=True, for max_features=sqrt(n_features) and max_features=n_features, respectively.

[11]:
```
np.random.seed(0)
rf_estimators10 = RandomForestClassifier(n_estimators=10, max_features=int(np.sqrt(X_train.shape[1])), bootstrap=True
                                         , criterion='gini', max_depth=None)
rf_estimators10.fit(X_train, y_train)
acc = accuracy_score(y_val, rf_estimators10.predict(X_val))
print(f'Q6-1 max_features= sqrt: {acc:.2f}')
# For Q6-1, validation accuracy should be higher than or equal to 0.88
```

```
Q6-1 max_features= sqrt: 0.88
```

[12]:
```
np.random.seed(0)
rf_estimators10 = RandomForestClassifier(n_estimators=10, max_features=None, bootstrap=True, criterion='gini', max_depth=None)
rf_estimators10.fit(X_train, y_train)
acc = accuracy_score(y_val, rf_estimators10.predict(X_val))
print(f'Q6-2 max_features= None: {acc:.2f}')
# For Q7-2, validation accuracy should be higher than or equal to 0.86
```

```
Q6-2 max_features= None: 0.88
```

7. Explain how you chose/design your model and what feature processing you have done in detail. Otherwise, no points will be given.
   - ClassifyNode: A class representing a node in a decision tree. Each node contains the best feature and threshold for splitting, left and right child nodes, and a class mark for leaf nodes.
   - MyDecisionTreeClassifier: A custom Decision Tree Classifier class that takes various parameters such as criterion (gini or entropy), min_samples_split, max_depth, n_feats, and randomSet. The class provides methods to fit the data, predict labels, and calculate feature importance.
   - RandomForestClassifier: A custom Random Forest Classifier class that takes parameters like n_estimators, max_features, bootstrap, criterion, max_depth, and randomSet. It leverages the custom MyDecisionTreeClassifier to create an ensemble of decision trees. The class provides methods to fit the data and predict labels.

     In the previously mentioned discussion, we observed that setting n_estimators to 50 yields an accuracy of 89%. It has been discovered that increasing this value to 80 can further improve the accuracy to 89.875%. Enabling the randomSet parameter allows the dataset to be trained in a random manner during the learning process. This helps Decision trees to learn more effectively and reduces the likelihood of overfitting.

Part. 2 Question (30%)

1. Answer the following questions in detail:
   A. Why does a decision tree tend to overfit the training set?
      - High Complexity: Decision trees can become very complex if they are allowed to grow without constraints. The tree's depth increases as it learns from the training set, attempting to capture even the smallest patterns or noise present in the data. This results in a highly complex model that performs well on the training set but fails to generalize to

new, unseen data.

- Lack of Regularization: Decision trees do not have any built-in regularization mechanism. Regularization is a technique used to reduce the complexity of a model and prevent overfitting. Without regularization, decision trees can easily become overly complex and overfit the training set.
- Greedy Splitting: Decision trees use a greedy approach for splitting the data at each node. This means that the tree is constructed by making the best decision at each step, without considering the long-term impact of the current split on future splits. This can lead to suboptimal trees that are highly specific to the training set.

B. Is it possible for a decision tree to achieve 100% accuracy on the training set?

- Yes, it is possible for a decision tree to achieve 100% accuracy on the training set, especially if the tree is allowed to grow without any constraints. The decision tree can create enough splits and branches to perfectly classify each instance in the training set. However, this is often a sign of overfitting, as the tree might not generalize well to unseen data.

C. List and describe at least three strategies we can use to reduce the risk of overfitting in a decision tree.

- Pruning: Pruning is a technique used to reduce the size and complexity of a decision tree by removing branches that do not contribute significantly to the overall performance of the tree. Two common pruning methods are pre-pruning and post-pruning. Pre-pruning involves stopping the tree growth early, while post-pruning involves growing the tree to its full size and then removing branches based on some criteria.
- Limit Tree Depth: By limiting the maximum depth of the decision tree, we can prevent it from becoming overly complex. This helps the tree generalize better to unseen data by ensuring it doesn't capture noise or irrelevant patterns from the training set.
- Ensemble Methods: Ensemble methods, such as random forests and gradient boosting machines, involve building multiple decision trees and combining their predictions. These methods help reduce overfitting by averaging the results of individual trees, thereby

reducing the impact of any single overfit tree on the final prediction.

- Feature Selection: Reducing the number of features used to build the decision tree can help prevent overfitting. Feature selection techniques, such as recursive feature elimination, can be used to identify the most relevant features for the task, thereby reducing the complexity of the tree and improving its generalization capabilities.
- Cross-validation: Cross-validation is a technique used to evaluate the performance of a model on unseen data. By dividing the dataset into multiple folds and training the model on different combinations of these folds, we can get an estimate of how well the model is likely to perform on new data. This can help us identify and prevent overfitting early in the model development process.

2. For each statement, answer True or False and provide a detailed explanation:

A. True. In AdaBoost, the weights of misclassified examples go up by the same multiplicative factor. This factor is determined by the performance of the current weak classifier. Specifically, the weights of misclassified examples are updated using the formula:
new_weight = old_weight * exp(alpha_t),
where alpha_t = 1/2 * ln((1 - error_rate) / error_rate),
and error_rate is the weighted error rate of the current weak classifier. Thus, all misclassified examples have their weights increased by the same multiplicative factor, exp(alpha_t).

B. False. In AdaBoost, the weighted training error (εt) of the tth weak classifier on training data with weights Dt tends to decrease as a function of t. The reason is that AdaBoost adjusts the weights of the training examples at each iteration, giving higher weights to misclassified instances. This makes it more likely for the next weak classifier to focus on these harder-to-classify examples and potentially perform better. As the algorithm iterates, it becomes more challenging for the weak classifiers to achieve high error rates on the increasingly difficult examples, so the weighted training error tends to decrease over time.

C. False. While AdaBoost can often achieve low training error, it is not guaranteed to achieve zero training error regardless of the type of weak classifier used or the number of iterations performed. The ability of AdaBoost to reach zero training error depends on the complexity of the problem, the quality of the weak classifiers, and the amount of noise in the

training data. If the weak classifiers are too simple or the problem is too complex, AdaBoost may not be able to find a strong classifier that can perfectly classify the training data, even with a large number of iterations. Additionally, if the training data contains noise or outliers, AdaBoost might not achieve zero training error, as the algorithm is sensitive to noisy data and outliers due to the adaptive nature of the weights.

3. This table displays the misclassification rate, cross-entropy, and Gini index for each leaf node in Tree A and Tree B.

Tree	Node	misclassification rates	cross-entropy	Gini index
A	1	200/600	0.9184	0.4444
A	2	0/200	0	0
B	1	100/400	0.8113	0.3750
B	2	100/400	0.8113	0.3750

This table shows the overall misclassification rate, cross-entropy, and Gini index for Tree A and Tree B.

Tree	misclassification rates	cross-entropy	Gini index
A	400/800	0.6888	0.3333
B	400/800	0.8113	0.3750