NYCU Pattern Recognition, Homework 2

Deadline: Apr. 26, 23:59

Part. 1, Coding (80%):

For this coding assignment, you are required to implement the Decision Tree and Random Forest algorithms using only NumPy. Afterward, you will need to train your model on the provided dataset and evaluate its performance on the validation data.

(30%) Decision Tree

Requirements:

- Implement the **Gini** and **Entropy** for measuring the "best" splitting of the data.
- Implement the Decision Tree algorithm (CART, Classification and Regression Trees) with the following 3 arguments:
- **Criterion**: The function to measure the quality of a split of the data. Your model should support "gini" for the Gini impurity and "entropy" for the information gain.
- **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 splitting data once.
- **Max_features**: The number of features to consider when looking for the best split. If None, then max_features=n_features.
- For more detailed descriptions of the arguments, please refer to Scikit-learn.
- 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 Entropy and Gini index of the array provided in the sample code, using the formulas on page 6 of the HW3 slide.

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

['+' '+' '+' '+' '+' '-']: gini index = 0.2777777777777778
['+' '+' '+' '-' '-' '-']: gini index = 0.5
['+' '-' '-' '-' '-' '-']: gini index = 0.2777777777777778
```

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.

```python
# For Q2-1, validation accuracy should be higher than or equal to 0.73

np.random.seed(0) # You may adjust the seed number in all the cells

dt_depth3 = DecisionTree(criterion='gini', max_features=None, max_depth=3)
dt_depth3.fit(X_train, y_train)

acc = accuracy_score(y_val, dt_depth3.predict(X_val))

print("Q2-1 max_depth=3: ", acc)
```

```
Q2-1 max_depth=3:  0.73125
```

```python
# For Q2-2, validation accuracy should be higher than or equal to 0.85

np.random.seed(0)

dt_depth10 = DecisionTree(criterion='gini', max_features=None, max_depth=10)
dt_depth10.fit(X_train, y_train)

print("Q2-2 max_depth=10: ", accuracy_score(y_val, dt_depth10.predict(X_val)))
```

```
Q2-2 max_depth=10:  0.865
```

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.

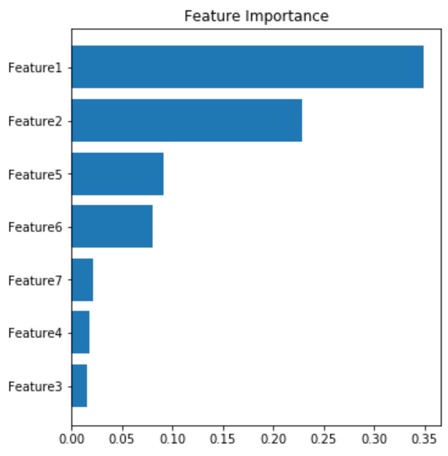
```python
# For Q3-1, validation accuracy should be higher than or equal to 0.73

np.random.seed(0)

dt_gini = DecisionTree(criterion='gini', max_features=None, max_depth=3)
dt_gini.fit(X_train, y_train)

print("Q3-1 criterion='gini': ", accuracy_score(y_val, dt_gini.predict(X_val)))
```

```
Q3-1 criterion='gini':  0.73125
```

```python
# For Q3-2, validation accuracy should be higher than or equal to 0.77

np.random.seed(0)

dt_entropy = DecisionTree(criterion='entropy', max_features=None, max_depth=3)
dt_entropy.fit(X_train, y_train)

print("Q3-2 criterion='entropy': ", accuracy_score(y_val, dt_entropy.predict(X_val)))
```

```
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.



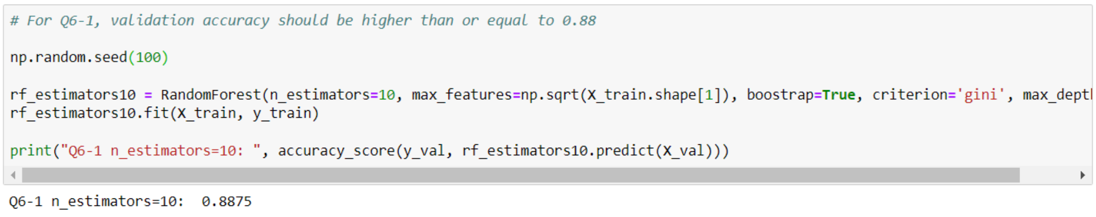
(20%) Random Forest

Requirements:

- Fix the random seed.
- Implement the Random Forest algorithm by using the CART you just implemented.
- The Random Forest model should include the following three arguments:
- **N_estimators**: The number of trees in the forest.
- **Max_features**: The number of features to consider when looking for the best split using the decision tree.
- **Bootstrap**: Whether to use bootstrap samples when building trees.
- For more detailed descriptions of the arguments, please refer to Scikit-learn.
- Use majority voting to obtain the final prediction.

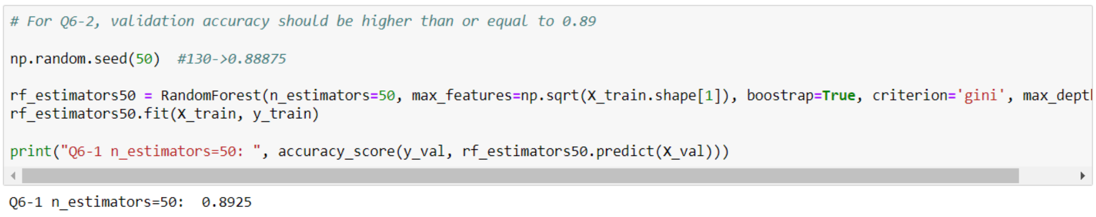
Criteria:

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.



```
# For Q6-1, validation accuracy should be higher than or equal to 0.88

np.random.seed(100)

rf_estimators10 = RandomForest(n_estimators=10, max_features=np.sqrt(X_train.shape[1]), boostrap=True, criterion='gini', max_dept
rf_estimators10.fit(X_train, y_train)

print("Q6-1 n_estimators=10: ", accuracy_score(y_val, rf_estimators10.predict(X_val)))
```

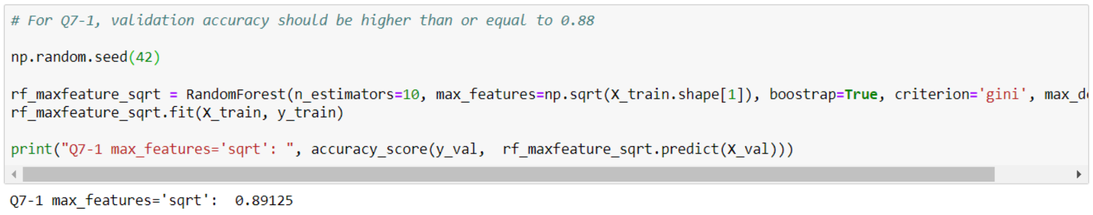
```
Q6-1 n_estimators=10:  0.8875
```



```
# For Q6-2, validation accuracy should be higher than or equal to 0.89

np.random.seed(50)  #130->0.88875

rf_estimators50 = RandomForest(n_estimators=50, max_features=np.sqrt(X_train.shape[1]), boostrap=True, criterion='gini', max_dept
rf_estimators50.fit(X_train, y_train)

print("Q6-1 n_estimators=50: ", accuracy_score(y_val, rf_estimators50.predict(X_val)))
```

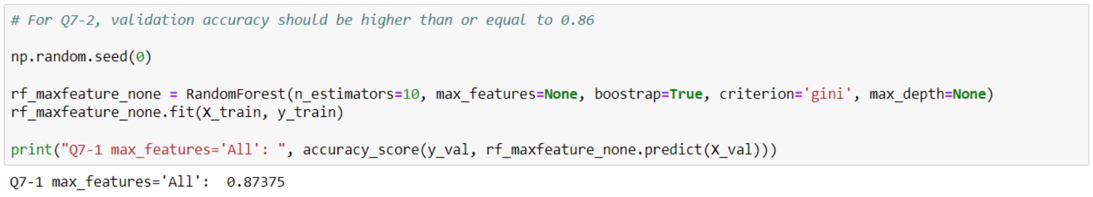
```
Q6-1 n_estimators=50:  0.8925
```

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.



```
# For Q7-1, validation accuracy should be higher than or equal to 0.88

np.random.seed(42)

rf_maxfeature_sqrt = RandomForest(n_estimators=10, max_features=np.sqrt(X_train.shape[1]), boostrap=True, criterion='gini', max_d
rf_maxfeature_sqrt.fit(X_train, y_train)

print("Q7-1 max_features='sqrt': ", accuracy_score(y_val,  rf_maxfeature_sqrt.predict(X_val)))
```

```
Q7-1 max_features='sqrt':  0.89125
```



```
# For Q7-2, validation accuracy should be higher than or equal to 0.86

np.random.seed(0)

rf_maxfeature_none = RandomForest(n_estimators=10, max_features=None, boostrap=True, criterion='gini', max_depth=None)
rf_maxfeature_none.fit(X_train, y_train)

print("Q7-1 max_features='All': ", accuracy_score(y_val, rf_maxfeature_none.predict(X_val)))
```

```
Q7-1 max_features='All':  0.87375
```

(20%) Train your own model

Requirements:

- Train your model (either Decision Tree or Random Forest).
- Try different parameters and feature engineering to beat the baseline.
- Save your test predictions in a CSV file.

Criteria:

7. (20%) Explain how you chose/design your model and what feature processing you have done in detail. Otherwise, no points will be given.

Points	Testing Accuracy
20 points	acc > 0.915
15 points	acc > 0.9
10 points	acc > 0.88
5 points	acc > 0.8
0 points	acc <= 0.8

First of all, I think that the setting of more estimators will get the better result, which means the more sub-tree in the Random Forest. But after experimenting, it proved that my idea was wrong, the reason I thought adding more estimators to a random forest model can improve its performance up to a certain point, but beyond that point, this setting will not necessarily lead to better results and may actually lead to overfitting. As the number of trees in the random forest increases, the variance of the model decreases, which means the model becomes more stable and less sensitive to noise in the data. As a result, the *n_estimators* of my implementation is set to 20, the settings of larger or smaller than 20 will give poor results.

For the feature engineering, I didn't do any other processing compared with the previous part in my final implementation. But in my experimental process, I plotted the correlations between all features and target, it is observed that the Feature1, Feature2 and Feature3 have the negative correlations with other variables. So, I tried to remove these three features and trained my model, this operation didn't bring to the expected results. On the other hand, I does not require standardization of the data before input the model, because Random Forest works by recursively partitioning the data into smaller subsets based on the values of individual features, and the resulting split is based on a threshold value that is specific to each feature. Standardizing the data would not change the ranking of the features or their split points, and therefore would not affect the outcome of the model.

Finally, I want to explain why I chose “entropy” as my *criterion*, when I use df_train.head() this code to visualize the data, I found that all features were continuous variables, and entropy is preferred for continuous variables, gini is preferred for categorical variables.

Part. 2, Questions (30%):

1. Answer the following questions in detail:
    a. Why does a decision tree tend to overfit the training set?
    I think there are several reasons:
        - Decision trees can be sensitive to noise and outliers in the data. This can result in a tree that is overly complex and does not generalize well to new data.
        - Decision trees do not have any inherent regularization, such as L1 or L2 penalties, to prevent overfitting.
        - Decision trees use a greedy search strategy to find the best split at each node, meaning that they choose the locally optimal split at each step without considering the global optimization. This can lead to a tree that is too complex and overfits the training data.

    b. Is it possible for a decision tree to achieve 100% accuracy on the training set?
    My answer is yes. If the tree is complex enough and the training set is not too noisy, it is possible for a decision tree to achieve 100% accuracy on the training set. In fact, it is relatively easy to construct a decision tree that perfectly fits the training data, simply by increasing the complexity of the tree until each leaf node contains only a single training example. However, such a tree is likely to be overfit to the training set and will not generalize well to new, unseen data.

    c. List and describe at least three strategies we can use to reduce the risk of overfitting in a decision tree.
        - Pruning: Removing branches or nodes that do not contribute significantly to the overall performance of the tree.
        - Regularization: Adding a penalty term to the objective function that is being optimized during the tree construction. This penalty term encourages

the tree to be less complex and reduces the risk of overfitting. For example, L1 regularization and L2 regularization.

- Ensemble methods: Bagging involves building multiple decision trees on different subsets of the data and combining their predictions to make the final prediction. Boosting involves building a sequence of decision trees, where each tree is trained on a modified version of the data that focuses on the examples that were misclassified by the previous tree.

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

   a. In AdaBoost, weights of the misclassified examples go up by the same multiplicative factor.

      The answer is "True" since that the AdaBoost algorithm actually do that. If your answer is "False", You should indicate that the multiplicative factors are different "in different iterations".

      False. In AdaBoost, the weights of the misclassified examples go up by different multiplicative factors, depending on their importance in the current iteration. Specifically, the weight of an example is increased if it was misclassified in the previous iteration, and the amount of increase depends on the error rate of the previous weak classifier.

   b. In AdaBoost, weighted training error $\boldsymbol{\varepsilon}_t$ of the t_{th} weak classifier on training data with weights D_t tends to increase as a function of t.

      The answer should be "True". During the boosting iterations, the weak classifiers are forced to classify more difficult examples, causing the weights to increase for the examples that are repeatedly misclassified. As a result, the ε_t, of the t-th weak classifier tends to increase.

      False. In AdaBoost, the weighted training error $\boldsymbol{\varepsilon}_t$ of the t_{th} weak classifier tends to decrease as a function of t, not increase. This is because AdaBoost puts more emphasis on examples that were misclassified by the previous weak classifier, making them more important for subsequent weak classifiers to learn from.

   c. AdaBoost will eventually give zero training error regardless of the type of weak classifier it uses, provided enough iterations are performed.

False. Although AdaBoost can reduce the training error to near zero after a sufficient number of iterations, it does not guarantee to give zero training error regardless of the type of weak classifier used.

3. Consider a data set comprising 400 data points from class C_1 and 400 data points from class C_2. Suppose that a tree model A splits these into (200, 400) at the first leaf node and (200, 0) at the second leaf node, where (n, m) denotes that n points are assigned to C_1 and m points are assigned to C_2. Similarly, suppose that a second tree model B splits them into (300, 100) and (100, 300). **Evaluate the <u>misclassification rates</u> for the two trees and hence show that they are equal**. Similarly, **evaluate the cross-entropy** $\boldsymbol{Entropy = -\sum_{k=1}^{K} p_k \log_2 p_k}$ and **Gini index** $Gini = 1 - \sum_{k=1}^{K} p_k^2$ **for the two trees**. Define p_k to be the proportion of data points in region R assigned to class k, where k = 1, . . . , K.

misclassification rate A

$= \frac{0+200}{400+400} = \frac{1}{4}$

misclassification rate B

$= \frac{100+100}{400+400} = \frac{1}{4}$

[2-3]
(partial correct)
(5 points) misclassification rate 0.25
(2 points)
Tree A CrossEntropy: 0.6888
Tree B CrossEntropy: 0.8113
(2 points)
Tree A Gini: 1/3
Tree B Gini: 3/8

(All correct) (10 points)