NYCU Pattern Recognition, Homework 3

Ivan K. 0860838

PART 1, CODING

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.
Output:
['+' '+' '+' '+' '+' '-']: entropy = 0.6500224187629642
['+' '+' '+' '-' '-' '-']: entropy = 0.99999999711461
['+' '-' '-' '-' '-' '-']: entropy = 0.6500224187629642

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

```
# For Q1
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")
```

✓ 0.0s

```
['+' '+' '+' '+' '+' '-']: entropy = 0.6500224187629642
['+' '+' '+' '-' '-' '-']: entropy = 0.99999999711461
['+' '-' '-' '-' '-' '-']: entropy = 0.6500224187629642

['+' '+' '+' '+' '+' '-']: 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.
Accuracy score for max_depth=3:

Q2-1 max_depth=3: 0.73125

```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, sample_weight=None)

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

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

✓ 0.7s Python

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

Accuracy score for max_depth=3:

Q2-1 max_depth=3: 0.86125

```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=10)
dt_depth3.fit(X_train, y_train, sample_weight=None)

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

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

✓ 1.2s Python

```
Q2-1 max_depth=3:  0.86125
```

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.

Accuracy score for gini:

Q3-1 criterion='gini': 0.73125

```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, sample_weight=None)

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

✓ 0.7s Python

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

Accuracy score for entropy:

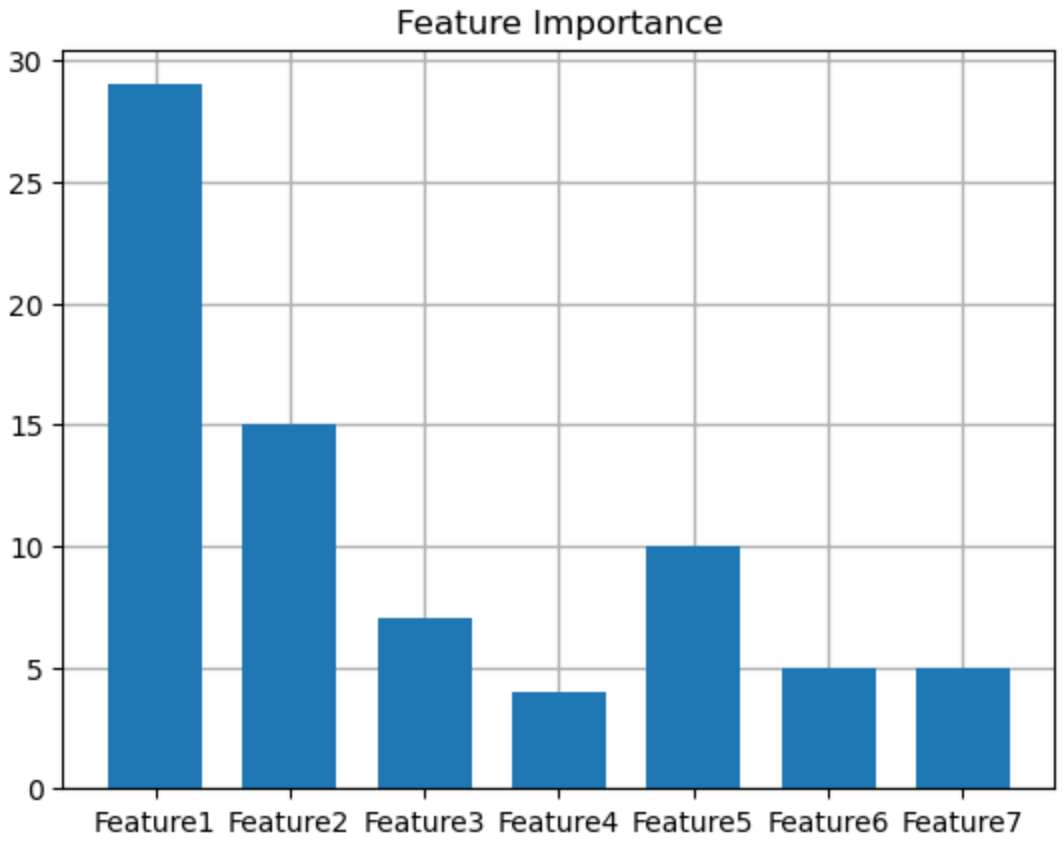
Q3-2 criterion='entropy': 0.77

```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, sample_weight=None)

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

✓ 0.6s

```
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.



Random Forest

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.

Q5-1 n_estimators=10: 0.8775

```
# For Q5-1, validation accuracy should be higher than or equal to 0.88

np.random.seed(0)

rf_estimators10 = RandomForest(n_estimators=10, max_features=np.sqrt(X_train.shape[1]), boostrap=True, criterion='gini', max_depth=None)
rf_estimators10.fit(X_train, y_train)

print("Q5-1 n_estimators=10: ", accuracy_score(y_val, rf_estimators10.predict(X_val)))
```

✓ 4.1s

```
Q5-1 n_estimators=10:  0.8775
```

Q5-1 n_estimators=50: 0.89625

```
# For Q5-2, validation accuracy should be higher than or equal to 0.89

np.random.seed(0)

rf_estimators50 = RandomForest(n_estimators=50, max_features=np.sqrt(X_train.shape[1]), boostrap=True, criterion='gini', max_depth=None)
rf_estimators50.fit(X_train, y_train)

print("Q5-1 n_estimators=50: ", accuracy_score(y_val, rf_estimators50.predict(X_val)))
```

✓ 19.6s

```
Q5-1 n_estimators=50:  0.89625
```

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.

Q6-1 max_features='sqrt': 0.8775

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

np.random.seed(0)

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

print("Q6-1 max_features='sqrt': ", accuracy_score(y_val,  rf_maxfeature_sqrt.predict(X_val)))
```

✓ 4.1s

```
Q6-1 max_features='sqrt':  0.8775
```

Q6-2 max_features='All': 0.87

```
# For Q6-2, 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("Q6-1 max_features='All': ", accuracy_score(y_val, rf_maxfeature_none.predict(X_val)))
```

✓ 4.6s

```
Q6-1 max_features='All':  0.87
```

Train your own model

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

```
np.random.seed(0)

model = RandomForest(n_estimators=50, max_features=4, boostrap=True, criterion='gini', max_depth=15)
model.fit(X_train, y_train)

print("Validation score: ", accuracy_score(y_val, model.predict(X_val)))
```

✓ 25.8s

```
Validation score:  0.90625
```

Validation score: 0.90625

Part. 2, Questions

1. Answer the following questions in detail:

a. Why does a decision tree tend to overfit the training set?

Decision tree tend to overfit the training set because if we don't specify depth of a tree each cut will be done until leaf will be pure, or in other words all classes in leaf will be the same class i.e. unique. So we train tree to fit our data specifically and this overfit our model.

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

Yes, if we just overfit our tree without limiting its max depth, we can reach 100% accuracy on the training dataset.

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

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

acc = accuracy_score(y_train, dt_depth3.predict(X_train))

print("Train acc: ", acc)
```

✓ 1.2s

```
Train acc:  1.0
```

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

Use max depth limit - limit depth of the tree.
Use pruning - one of the approaches is to replace subtree by a leaf.
Use minimum size for leaf - minimum number of instances that used to compose the leaf.
Use minimum size for a split - minimum number of instances that needed to make a split.

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.

True, as you can see from update equation

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

$D_t(i) = 1/m$, same multiplicative factor.

b. In AdaBoost, weighted training error εt of the the weak classifier on training data with weights Dt tends to increase as a function of t.

True. As training happens weak classifiers are trying to classify more difficult examples. Weight for misclassified examples will be increased. So the weighted training error of t_{th} weak classifier will increase.

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

False, it doesn't matter how many iterations are performed, zero error will never be achieved. If for example non-linear data is not well separated by a combination of linear classifiers that are 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 misclassification rates for the two trees and hence show that they are equal**. Similarly, **evaluate the cross-entropy** $Entropy = -\sum_{k=1}^{K} p_k log_2 p_k$ 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.

As you can see we have the same split for trees A and B (0,1) and (1,0) so given that we only make classification on classes we will have equal misclassification rates. Also we can see gini index and entropy for leaf nodes.

A 1 0
B 0 1
A gini: 0.4444444444444444 0.0
B gini: 0.375 0.375
A entropy: 0.9182958311690995 -1.4426951595367387e-09
B entropy: 0.8112781215737428 0.8112781215737428

```python
import numpy as np
C1 = np.zeros(400)
C2 = np.zeros(400) +1

# Tree model A
Aleaf_11 = (C1[:200], C2)
Aleaf_12 = (C1[:200], C1[:0])

# Tree model B
Bleaf_11 = (C1[:300], C2[:100])
Bleaf_12 = (C1[:100], C2[:300])

Aleaf_11_c = np.concatenate(Aleaf_11)
Aleaf_12_c = np.concatenate(Aleaf_12)

Aleaf_11_class = np.argmax(np.bincount(Aleaf_11_c.astype(int)))
Aleaf_12_class = np.argmax(np.bincount(Aleaf_12_c.astype(int)))
print('A', Aleaf_11_class, Aleaf_12_class)

Bleaf_11_c = np.concatenate(Bleaf_11)
Bleaf_12_c = np.concatenate(Bleaf_12)

Bleaf_11_class = np.argmax(np.bincount(Bleaf_11_c.astype(int)))
Bleaf_12_class = np.argmax(np.bincount(Bleaf_12_c.astype(int)))
print('B', Bleaf_11_class, Bleaf_12_class)
#################################################################
print('A gini:', gini(Aleaf_11_c), gini(Aleaf_12_c))
print('B gini:', gini(Bleaf_11_c), gini(Bleaf_12_c))

print('A entropy:', entropy(Aleaf_11_c), entropy(Aleaf_12_c))
print('B entropy:', entropy(Bleaf_11_c), entropy(Bleaf_12_c))
```

✓ 0.0s

```
A 1 0
B 0 1
A gini: 0.4444444444444444 0.0
B gini: 0.375 0.375
A entropy: 0.9182958311690995 -1.4426951595367387e-09
B entropy: 0.8112781215737428 0.8112781215737428
```