NYCU Pattern Recognition, Homework 2

311552034, 林柏翰

Part. 1, Coding (70%):

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.

```
# 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")
```

✓ 0.2s

```
['+' '+' '+' '+' '+' '-']: 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.

```
# 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)
```

✓ 0.6s

```
Q2-1 max_depth=3:  0.7325
```

```
# 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, sample_weight=None)

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

✓ 1.3s

```
Q2-2 max_depth=10:  0.8525
```

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.

```
# 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)))
```

✓ 0.5s

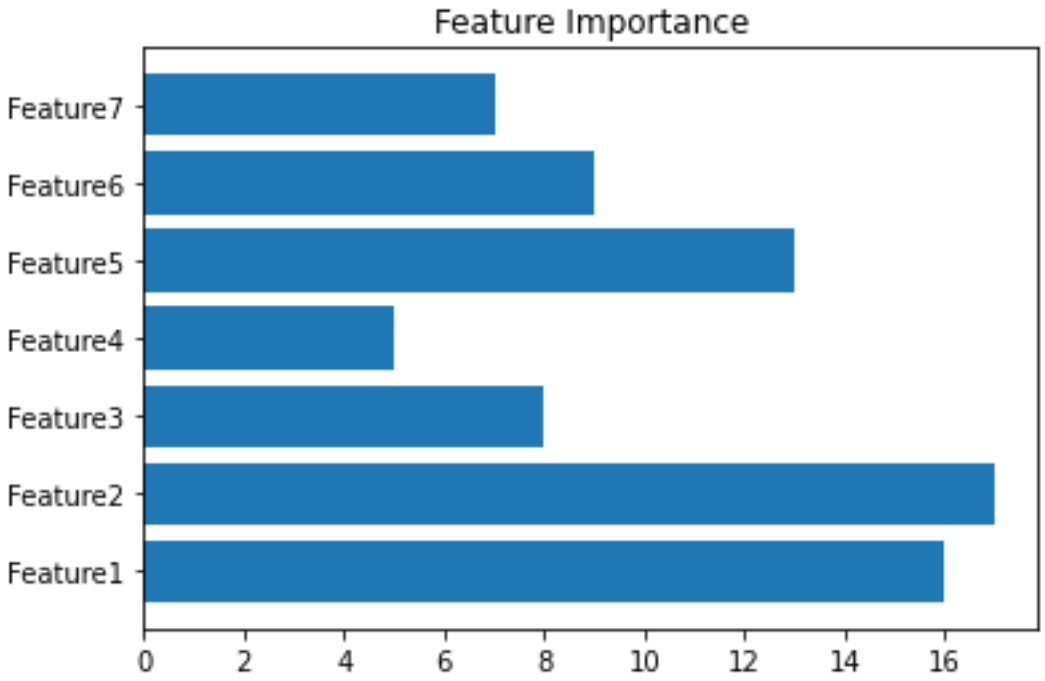
```
Q3-1 criterion='gini':  0.7325
```

```
# 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.



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 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', ma
rf_estimators10.fit(X_train, y_train)

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

✓ 3.1s Pyth

```
Q6-1 n_estimators=10:  0.88625
```

```
# 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_
rf_estimators50.fit(X_train, y_train)

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

✓ 15.4s Python

```
Q6-1 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.

```
# 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='gin
rf_maxfeature_sqrt.fit(X_train, y_train)

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

✓ 3.1s P

```
Q7-1 max_features='sqrt':  0.88625
```

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

✓ 9.4s

```
Q7-1 max_features='All':  0.86625
```

7. **Train your own model**

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

Ans:

Just like HW1 and HW2, I decided to use polynomial feature expansion to create feature.

According to the feature importance graph. I decided to skip the expansion of feature 4&7.

```python
def poly_expand(dataframe):
    dataframe['Feature1*50']=dataframe['Feature1']*50
    dataframe['Feature2*50']=dataframe['Feature2']*50
    dataframe['Feature1^2']=dataframe['Feature1']**2
    dataframe['Feature1 Feature2']=dataframe['Feature1']*dataframe['Feature2']
    dataframe['Feature1 Feature3']=dataframe['Feature1']*dataframe['Feature3']
    # dataframe['Feature1 Feature4']=dataframe['Feature1']*dataframe['Feature4']
    dataframe['Feature1 Feature5']=dataframe['Feature1']*dataframe['Feature5']
    dataframe['Feature1 Feature6']=dataframe['Feature1']*dataframe['Feature6']
    # dataframe['Feature1 Feature7']=dataframe['Feature1']*dataframe['Feature7']
    dataframe['Feature2^2']=dataframe['Feature2']**2
    dataframe['Feature2 Feature3']=dataframe['Feature2']*dataframe['Feature3']
    # dataframe['Feature2 Feature4']=dataframe['Feature2']*dataframe['Feature4']
    dataframe['Feature2 Feature5']=dataframe['Feature2']*dataframe['Feature5']
    dataframe['Feature2 Feature6']=dataframe['Feature2']*dataframe['Feature6']
    # dataframe['Feature2 Feature7']=dataframe['Feature2']*dataframe['Feature7']
    dataframe['Feature3^2']=dataframe['Feature3']**2
    # dataframe['Feature3 Feature4']=dataframe['Feature3']*dataframe['Feature4']
    dataframe['Feature3 Feature5']=dataframe['Feature3']*dataframe['Feature5']
    dataframe['Feature3 Feature6']=dataframe['Feature3']*dataframe['Feature6']
    # dataframe['Feature3 Feature7']=dataframe['Feature3']*dataframe['Feature7']
    # dataframe['Feature4^2']=dataframe['Feature4']**2
    # dataframe['Feature4 Feature5']=dataframe['Feature4']*dataframe['Feature5']
    # dataframe['Feature4 Feature6']=dataframe['Feature4']*dataframe['Feature6']
    # dataframe['Feature4 Feature7']=dataframe['Feature4']*dataframe['Feature7']
    dataframe['Feature5^2']=dataframe['Feature5']**2
    dataframe['Feature5 Feature6']=dataframe['Feature5']*dataframe['Feature6']
    # dataframe['Feature5 Feature7']=dataframe['Feature5']*dataframe['Feature7']
    dataframe['Feature6^2']=dataframe['Feature6']**2
    # dataframe['Feature6 Feature7']=dataframe['Feature6']*dataframe['Feature7']
    # dataframe['Feature7^2']=dataframe['Feature7']**2
    return dataframe
```

After creating total 24 features, I decide to use random forest to train my model. To figure Out how many estimators I will need to obtain best accuracy, I use brute method to try many different number of estimator and the result shows that the model reached its best performance with 560 estimators.

```
Estimators: 200 , acc:  0.89875
Best renew :  200  , acc:  0.89875
Estimators: 220 , acc:  0.9
Best renew :  220  , acc:  0.9
Estimators: 240 , acc:  0.90125
Best renew :  240  , acc:  0.90125
Estimators: 260 , acc:  0.9
Estimators: 280 , acc:  0.90125
Estimators: 300 , acc:  0.90125
Estimators: 320 , acc:  0.90125
Estimators: 340 , acc:  0.9
Estimators: 360 , acc:  0.90125
Estimators: 380 , acc:  0.90125
Estimators: 400 , acc:  0.90125
Estimators: 420 , acc:  0.90125
Estimators: 440 , acc:  0.9025
Best renew :  440  , acc:  0.9025
Estimators: 460 , acc:  0.90125
Estimators: 480 , acc:  0.90125
Estimators: 500 , acc:  0.9
Estimators: 520 , acc:  0.90125
Estimators: 540 , acc:  0.90125
Estimators: 560 , acc:  0.90375
Best renew :  560  , acc:  0.90375
Estimators: 580 , acc:  0.9025
highest acc:  0.90375  with estimators= 560
```

and the best accuracy I got on validation set is 0.90375. Besides, I decided to use random seed 2 because 2 is my lucky number.

Part. 2, Questions (30%):

I. Answer the following question in detail:

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

A decision tree tends to overfit the training set because it can create a complex decision boundary that perfectly fits the training data. As a result, the model may memorize the training data, and fail to generalize well to new, unseen data. This is because the decision tree may split the data too finely, and include features that are not relevant to the target variable. Additionally, decision trees are sensitive to small changes in the training data, which can lead to different decision trees being generated for each set of training data. Overfitting can also occur when the decision tree is too deep, resulting in overly specific rules and decisions that only apply to the training data but not to new data.

B. Is it possible for a decision tree to achieve 100% accuracy on the training set?
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 or pruning. However, such a tree may not generalize well to new data. That is, such a model is likely to overfit the data and not generalize well to new, unseen data. And therefore may not perform as well on the test or validation set.

C. List and describe at least three strategies we can use to reduce the risk of overfitting in a decision tree.
   1. Pruning: Pruning is a technique used to simplify a decision tree by removing some of its branches or nodes that do not add much to its overall predictive power. This can help to prevent overfitting by reducing the complexity of the tree and making it more generalizable to new data.
   2. Setting constraints: Setting constraints such as minimum sample size per leaf node, maximum depth of the tree, and minimum information gain can help to prevent overfitting. These constraints limit the complexity of the tree, forcing it to focus on the most important features and relationships in the data.

   3. Ensemble methods: Ensemble methods such as Random Forest and Gradient Boosted Trees can also help to reduce the risk of overfitting. These methods use multiple decision trees to make predictions, each with a different subset of features and data samples. By combining the predictions of multiple trees, the ensemble can reduce the risk of overfitting while still achieving high accuracy on both the training and test sets.

II. 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, In AdaBoost, the weights of the misclassified examples are increased by the same multiplicative factor, which is given by the formula:

$$w\_i' = w\_i * \exp(\alpha)$$

where w_i is the weight of the ith example in the current iteration, w_i' is the updated weight of the ith example in the next iteration, and α is the weight of the current weak classifier.This formula ensures that the misclassified examples are given higher weights than the correctly classified examples, and the subsequent weak classifiers focus more on these examples in the next iteration. This process is repeated iteratively, with each iteration producing a new weak classifier and updated weights for the training examples.

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.

True ,In the course of boosting iterations the weak classifiers are forced to try to classify more difficult examples. The weights will increase for examples that are repeatedly misclassified by the weak classifiers. The weighted training error ϱ_t of the t^{th} weak classifier on the training data therefore tends to increase.

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

False, not true if the data in the training set cannot be separated by a linear combination of the specific type of weak classifiers we are using .

While the AdaBoost algorithm is designed to improve the classification accuracy of the weak classifiers, it is possible that some examples may be impossible to classify correctly, even with an infinite number of iterations. In such cases, the training error of the AdaBoost ensemble may reach a minimum nonzero value, and cannot bereduced any further.

III. 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** $\mathbf{Entropy= -\sum_{k=1}^{k} p_k log_2 p_k}$ and **Gini index** $\text{Gini}=1-\sum_{k=1}^{k} p_k^2$ **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.

model A

leaf node 1: $p = 200/600$, $q = 400/600$

entropy $= -\frac{1}{3}\log(\frac{1}{3}) - \frac{2}{3}\log(\frac{2}{3}) \approx 0.9183$

Gini $= 1 - (\frac{1}{9} + \frac{4}{9}) = \frac{4}{9} \doteq 0.44444$

miss $= \frac{200}{800} = 0.25$ #

leaf node 2: $p = 1$ $q = 0$

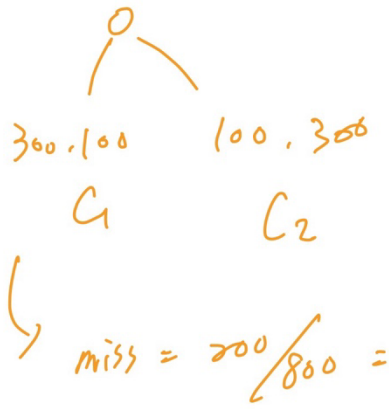
entropy $= -1\log_2 1 - 0\log_2 0 = 0$

Gini $= 1 - (1^2 + 0^2) = 0$

weighted avg. cross entropy $= 0.9183 \times \frac{3}{4} + 0 \times \frac{1}{4} \doteq 0.6887$ #

weighted avg. Gini $= 0.44444 \times \frac{3}{4} + 0 \times \frac{1}{4} = 0.3333$ #

similarly, for model B



#

weighted entropy = $\frac{1}{2}\left(\frac{3}{4}\log\frac{3}{4} + \frac{1}{4}\log\frac{1}{4}\right)$

$+$

$\frac{1}{2}\left(\frac{1}{4}\log\frac{1}{4} + \frac{3}{4}\log\frac{3}{4}\right)$

$= 2 - \frac{3}{4}\log_2 3$ #

$\doteq 0.811$

weighted Gini = $\frac{1}{2}\left\{1 - \left[\left(\frac{3}{4}\right)^2 + \left(\frac{1}{4}\right)^2\right]\right\}$

$+$

$\frac{1}{2}\left\{1 - \left[\left(\frac{1}{4}\right)^2 + \left(\frac{3}{4}\right)^2\right]\right\}$

$= \frac{3}{8}$ # $= 0.375$