HW3

Part 1:

Q1:

```
+ 程式碼   + M

# 1 = class 1,
# 2 = class 2
data = np.array([1,2,1,1,1,1,2,2,1,1,2])
✓ 0.9s

print("Gini of data is ", gini(data))
✓ 0.1s
Gini of data is  0.4628099173553719

print("Entropy of data is ", entropy(data))
✓ 0.9s
Entropy of data is  0.9456603046006401
```

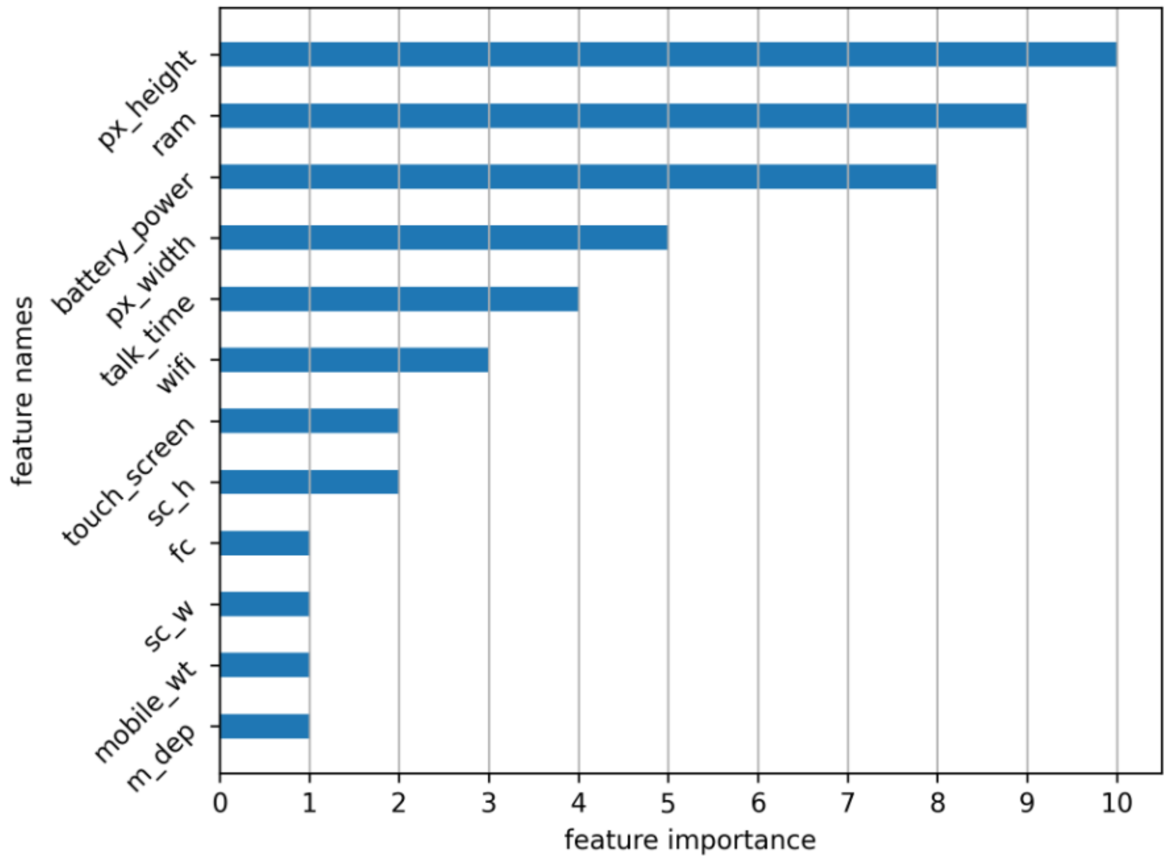
Q2:

```
print('Decision Tree')
clf_depth3 = DecisionTree(criterion='gini', max_depth=3)
clf_depth3.fit(x_data=x_train, y_data=y_train)
clf_depth3.get_feature_count()
pred = clf_depth3.predict(x_data=x_test)
print(f'accuracy_score: {accuracy_score(y_test, pred)}')
# clf_depth3.print_acc(ans=y_test, pred=pred)

clf_depth10 = DecisionTree(criterion='gini', max_depth=10)
clf_depth10.fit(x_data=x_train, y_data=y_train)
clf_depth10.get_feature_count()
pred = clf_depth10.predict(x_data=x_test)
print(f'accuracy_score: {accuracy_score(y_test, pred)}')
# clf_depth10.print_acc(ans=y_test, pred=pred)
✓ 23.8s                                                    Python

Decision Tree
{'ram': 3, 'battery_power': 3, 'px_height': 1}
accuracy_score: 0.92
{'ram': 9, 'battery_power': 8, 'px_height': 10, 'talk_time': 4, 'sc_h': 2, 'wifi': 3, 'px_width': 5, 'm_dep':
1, 'mobile_wt': 1, 'touch_screen': 2, 'sc_w': 1, 'fc': 1}
accuracy_score: 0.9433333333333334
```

Q3:



Q4:

Show the accuracy score of validation data by `n_estimators=10` and `n_estimators=100`, respectively.

```python
print('Adaboost')
ada_10est = AdaBoost(n_estimators=10)
ada_10est.fit(x_data=x_train, y_data=y_train)
pred = ada_10est.predict(x_data=x_test)
print(f'accuracy_score: {accuracy_score(y_test, pred)}')
# ada_10est.print_acc(y_test, pred)

ada_100est = AdaBoost(n_estimators=100)
ada_100est.fit(x_data=x_train, y_data=y_train)
pred = ada_100est.predict(x_data=x_test)
print(f'accuracy_score: {accuracy_score(y_test, pred)}')
# # ada_100est.print_acc(y_test, pred)
```

✓ 7m 10.3s

```
Adaboost
accuracy_score: 0.95
accuracy_score: 0.9766666666666667
```

Q5-1:

- 上面是n_estimator=100，下面是n_estimator=10。

Using `criterion=gini`, `max_depth=None`, `max_features=sqrt(n_features)`, showing the accuracy score of validation data by `n_estimators=10` and `n_estimators=100`, respectively.

```python
print('Random Forest()')
clf_100tree = RandomForest(n_estimators=100, max_features=np.sqrt(x_train.shape[1]), max_depth=None, crite
clf_100tree.fit(x_data=x_train, y_data=y_train)
pred = clf_100tree.predict(x_data=x_test)
print(f'accuracy_score: {accuracy_score(y_test, pred)}')
# # clf_100tree.print_acc(y_test, pred)

clf_10tree = RandomForest(n_estimators=10, max_features=np.sqrt(x_train.shape[1]), max_depth=None, criteri
clf_10tree.fit(x_data=x_train, y_data=y_train)
pred = clf_10tree.predict(x_data=x_test)
print(f'accuracy_score: {accuracy_score(y_test, pred)}')
# clf_10tree.print_acc(y_test, pred)

```

[81] ✓ 6m 11.4s Python

```
Random Forest()
accuracy_score: 0.94
accuracy_score: 0.9433333333333334
```

Q5-2:

Question 5.2

Using `criterion=gini`, `max_depth=None`, `n_estimators=10`, showing the accuracy score of validation data by `max_features=sqrt(n_features)` and `max_features=n_features`, respectively.

```python
print('Random Forest-2()')
clf_random_features = RandomForest(n_estimators=10, max_features=np.sqrt(x_train.shape[1]))
clf_random_features.fit(x_data=x_train, y_data=y_train)
pred = clf_random_features.predict(x_data=x_test)
print(f'accuracy_score: {accuracy_score(y_test, pred)}')
# clf_random_features.print_acc(y_test, pred)

clf_all_features = RandomForest(n_estimators=10, max_features=x_train.shape[1])
clf_all_features.fit(x_data=x_train, y_data=y_train)
pred = clf_all_features.predict(x_data=x_test)
print(f'accuracy_score: {accuracy_score(y_test, pred)}')
# clf_all_features.print_acc(y_test, pred)
```

Python

```
Random Forest-2()
accuracy_score: 0.9233333333333333
accuracy_score: 0.9633333333333334
```

Q6:

- 我直接拿adaboost來用，n_estimator用30(因為發現後面的gain都沒變，可能error已經到0.5了)

```
def train_your_model(data):
    x_train = data.drop(labels=["price_range"], axis="columns")
    feature_names = x_train.columns.values
    x_train = x_train.values
    y_train = data['price_range'].values

    print("x_train:", type(x_train))
    print("y_train:", type(y_train))
    ## Define your model and training
    ada_35est = AdaBoost(n_estimators=30)
    ada_35est.fit(x_data=x_train, y_data=y_train)
    return ada_35est
```

✓ 0.3s

```
my_model = train_your_model(train_df)
```

✓ 1m 36.2s

```
x_train: <class 'numpy.ndarray'>
y_train: <class 'numpy.ndarray'>
```

+ 程式碼 + Markdown

```
y_pred = my_model.predict(x_test)
print(f'accuracy_score: {accuracy_score(y_test, y_pred)}')
```

✓ 0.1s

```
accuracy_score: 0.9766666666666667
```

Part 2:

Q1 :

- Decision Tree:
    - Contains `decision node` and `leaf node`
        - Decision node contains a question and lead to another node.
        - Leaf nodes are the answers, and we make a decision based on them
- Overfitting? 100% accuracy?
    - Since we could generate a tree that fits all the data by keep asking question until a leaf node refers to a specific situation(i.e. 1 data in each leaf node), it's always possible to have overfitting problem.
    - Even if we set some terminating condition, it's always possible that the tree overfits the data when your conditions are not strong enough to end the tree generation.
- How to reduce the risk of overfitting of a decision tree?
    - We could set some criteria and stop generate new leaf nodes when any of them are met. For example,
        - When the data set is pure → all data in a node falls into the same category.
        - Maximum tree depth is reached.
        - When a leaf node has too less data.
        - The **information gain** is less than a threshold.
    - **Bagging(Bootstrap Aggregation)**
        - Generate multiple decision trees and use them to get an aggregated predictor.
        - Each tree are contructed by analyzing **N** random samples**(with replacement)** from the initial dataset.
    - **Random forest**
        - A method specifically designed for decision tree.
        - In addition to **Bagging,** this methoid choose only *m* random attributes to make a decision at each node→improves randomness.

Q2 :

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

   - True.
   - The distribution at each iteration `t` is :

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

   - where $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$ and $\epsilon_j = \sum_{i=1}^{m} D_t(i)[y_i \neq h_j(x_i)]$
   - For wrongly classified data, $\frac{D_{t+1}(i)}{D_t(i)} = \frac{\exp[\alpha_t]}{Z_t}$, the ratio is the same for all wrongly classified date.

2. In AdaBoost, weighted training error ϵ_t of the t-th weak classifier on training data with weights D_t tends to increase as a function of t.

   - True.
   - We have $h_t = \arg\min_{h_j} \epsilon_j$. Since at each iteration, the weight for wrongly classfied data will increase. Those who are repeatedly being misclassfied(i.e. difficult examples) will tend to increase the training error as t increases.

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

   - False when the data are not linealy separable using the weak learner you define.(i.e. the training set cannot be separated by a linear combination of the weak classifiers.)
   - True if the data is linearly separable using the defined weak learner.

Q3 :

- Tree A: first leaf node is class 2 and second leaf node is class 1
  - # incorrect predictions = 200, # total predictions = 800
  - Misclassification Rate = 25%
- Tree B: first leaf node is class 1 and second leaf node is class 2
  - # incorrect predictions = (100+100), # total predictions = 800
  - Misclassification Rate = 25%
- They are equal

- Tree A
  - node1:
    - $p_1 = \frac{1}{3}, p_2 = \frac{2}{3}$
    - $entropy = -(p_1 \lg p_1 + p_2 \lg p_2) = 0.9183$
    - $gini = 1 - (p_1)^2 - (p_2)^2 = \frac{4}{9}$
  - node2:
    - $p_1 = 1, p_2 = 0$
    - $entropy = -(p_1 \lg p_1 + p_2 \lg p_2) = 0$
    - $gini = 1 - (p_1)^2 - (p_2)^2 = 0$
  - $CE = (\frac{3}{4}) \times 0.9183 + (\frac{1}{4}) \times 0 = 0.6887$
  - $Gini = \frac{3}{4} \times \frac{4}{9} = \frac{1}{3}$

- Tree B
  - node1:
    - $p_1 = \frac{3}{4}, p_2 = \frac{1}{4}$
    - $entropy = -(p_1 \lg p_1 + p_2 \lg p_2) = 0.8113$
    - $gini = 1 - (p_1)^2 - (p_2)^2 = \frac{3}{8}$
  - node2:
    - $p_1 = \frac{1}{4}, p_2 = \frac{3}{4}$
    - $entropy = -(p_1 \lg p_1 + p_2 \lg p_2) = 0.8113$
    - $gini = 1 - (p_1)^2 - (p_2)^2 = \frac{3}{8}$
  - $CE = \frac{1}{2} \times 0.8113 \times 2 = 0.8113$
  - $Gini = \frac{1}{2} \times \frac{3}{8} \times 2 = \frac{3}{8}$

- Since both Cross-Entropy and Gini of split A are smaller than those of split B, we choose split A over split B.