NYCU Introduction to Machine Learning, Homework 3

109550017黃品云

Part. 1, Coding (80%):

1. (5%) Gini Index or Entropy is often used for measuring the “best” splitting of the data. Please compute the Entropy and Gini Index of this array np.array([1,2,1,1,1,1,2,2,1,1,2]) by the for mula below. (More details on page 5 of the hw3 slides, 1 and 2 represent class1 and class 2, respectively)

```
print("Gini of data is ", gini(data))
```

```
Gini of data is  0.4628099173553719
```

```
print("Entropy of data is ", entropy(data))
```

```
Entropy of data is  0.9456603046006401
```

2. (10%) Implement the Decision Tree algorithm (CART, Classification and Regression Trees) and train the model by the given arguments, and print the accuracy score on the test data.

   2.1. Using Criterion= ‘gini’, showing the accuracy score of test data by Max_depth= 3 and Max_depth=10, respectively.

```
clf_depth3 = DecisionTree(criterion='gini', max_depth=3)
clf_depth10 = DecisionTree(criterion='gini', max_depth=10)
```

```
clf_depth3.fit(x_train, y_train)
print("criterion=gini, max_depth=3, accuracy:", accuracy_score(clf_depth3.predict(x_test), y_test))

clf_depth10.fit(x_train, y_train)
print("criterion=gini, max_depth=10, accuracy:", accuracy_score(clf_depth10.predict(x_test), y_test))
```

```
criterion=gini, max_depth=3, accuracy: 0.92
criterion=gini, max_depth=10, accuracy: 0.93
```

   2.2. Using Max_depth=3, showing the accuracy score of test data by Criterion= ‘gini’ and Criterion=’ entropy’, respectively.

```
clf_gini = DecisionTree(criterion='gini', max_depth=3)
clf_entropy = DecisionTree(criterion='entropy', max_depth=3)
```

```
clf_gini.fit(x_train, y_train)
print("criterion=gini, max_depth=3, accuracy:", accuracy_score(clf_gini.predict(x_test), y_test))

clf_entropy.fit(x_train, y_train)
print("criterion=entropy, max_depth=3, accuracy:", accuracy_score(clf_entropy.predict(x_test), y_test))
```

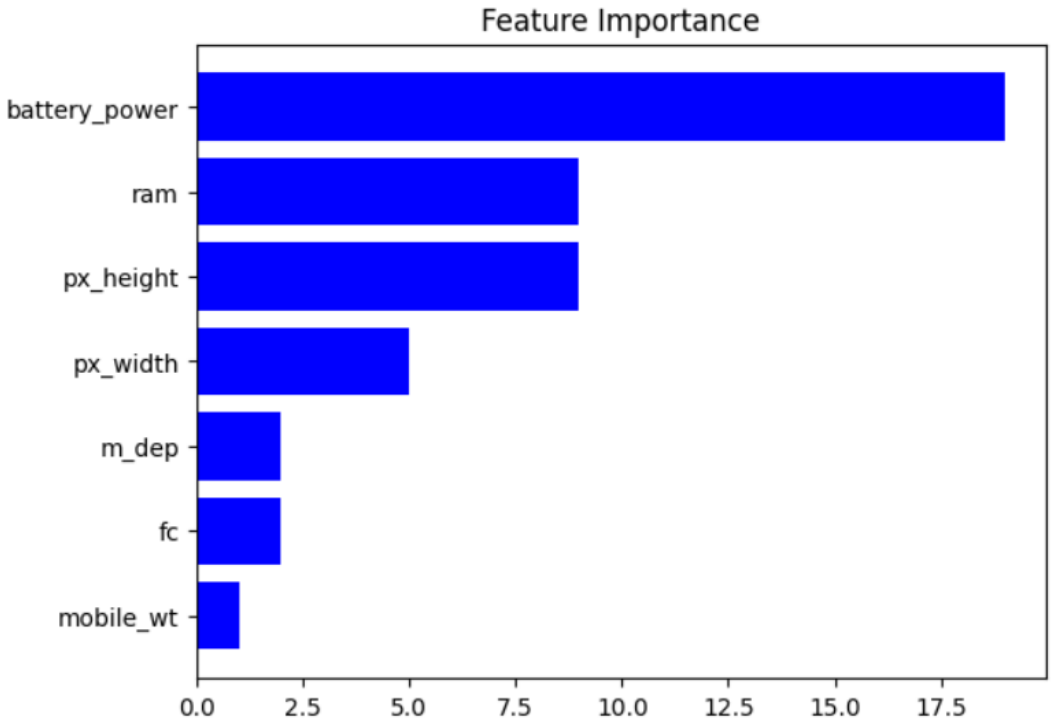
```
criterion=gini, max_depth=3, accuracy: 0.92
criterion=entropy, max_depth=3, accuracy: 0.9333333333333333
```

Note: Your decision tree scores should be over 0.9. It may suffer from overfitting, if so, you can tune the hyperparameter such as `max_depth`

Note: You should get the same results when re-building the model with the same arguments, ***no need to prune the trees***

Note: You can find the best split threshold by both methods. First one: 1) Try N-1 threshold values, where the i-th threshold is the average of the i-th and (i+1)-th sorted values. Second one: Use the unique sorted value of the feature as the threshold to split Hint: You can use the recursive method to build the nodes

3. (5%) Plot the feature importance of your Decision Tree model. You can use the model fro m Question 2.1, max_depth=10. (You can use simply counting to get the feature importanc e instead of the formula in the reference, more details on the sample code. Matplotlib is all owed to be used)



4. (15%) Implement the AdaBoost algorithm by using the CART you just implemented from question 2. You should implement one argument for the AdaBoost.
   1) N_estimators: The number of trees in the forest.
   4.1. Showing the accuracy score of test data by n_estimators=10 and n_estimators=100, respectively.

```
clf_Ada_10 = AdaBoost(n_estimators=10)
clf_Ada_10.fit(x_train, y_train)
y_pred = clf_Ada_10.predict(x_test)
```

```
print('n_estimators=10')
print('Test-set accuarcy score: ', accuracy_score(y_test, y_pred))
```

```
n_estimators=10
Test-set accuarcy score:  0.95
```

```
clf_Ada_100 = AdaBoost(n_estimators=100)
clf_Ada_100.fit(x_train, y_train)
y_pred = clf_Ada_100.predict(x_test)
```

```
print('n_estimators=100')
print('Test-set accuarcy score: ', accuracy_score(y_test, y_pred))
```

```
n_estimators=100
Test-set accuarcy score:  0.9766666666666667
```

5. (15%) Implement the Random Forest algorithm by using the CART you just implemented from question 2. You should implement three arguments for the Random Forest.
   1) N_estimators: The number of trees in the forest.
   2) Max_features: The number of features to consider when looking for the best split
   3) Bootstrap: Whether bootstrap samples are used when building trees

5.1. Using Criterion= 'gini' , Max_depth=None, Max_features=sqrt(n_features), Bootstrap=True, showing the accuracy score of test data by n_estimators=10 and n_estimators=100, respectively.

```
clf_10tree = RandomForest(n_estimators=10, max_features=np.sqrt(x_train.shape[1]))
clf_10tree.fit(x_train, y_train)
y_pred = clf_10tree.predict(x_test)
print('n_estimators=10, test-set accuarcy score: ', accuracy_score(y_test, y_pred))
```

```
n_estimators=10, test-set accuarcy score:  0.93
```

```
clf_100tree = RandomForest(n_estimators=100, max_features=np.sqrt(x_train.shape[1]))
clf_100tree.fit(x_train, y_train)
y_pred = clf_100tree.predict(x_test)
print('n_estimators=100, test-set accuarcy score: ', accuracy_score(y_test, y_pred))
```

```
n_estimators=100, test-set accuarcy score:  0.94
```

5.2. Using Criterion= 'gini' , Max_depth=None, N_estimators=10, Bootstrap=True, showing the accuracy score of test data by Max_features=sqrt(n_features) and Max_features=n_features, respectively.

Note: Use majority votes to get the final prediction, you may get different results when re-building the random forest model

```
clf_random_features = RandomForest(n_estimators=10, max_features=np.sqrt(x_train.shape[1]))
clf_all_features = RandomForest(n_estimators=10, max_features=x_train.shape[1])

clf_random_features.fit(x_train, y_train)
print("n_estimators=10, max_features=sqrt(n_features), accuracy:", accuracy_score(clf_random_features.predict(x_test), y_test

clf_all_features.fit(x_train, y_train)
print("n_estimators=10, max_features=n_features, accuracy:", accuracy_score(clf_all_features.predict(x_test), y_test))
```

```
n_estimators=10, max_features=sqrt(n_features), accuracy: 0.9366666666666666
n_estimators=10, max_features=n_features, accuracy: 0.9666666666666667
```

6. (20%) Tune the hyperparameter, perform feature engineering or implement more powerful ensemble methods to get a higher accuracy score. Screenshot your tests score on the report. Please note that only the ensemble method can be used. The neural network method is not allowed.

```
def train_your_model(x, y):
    ## Define your model and training
    my_model = AdaBoost(n_estimators=150)
    my_model.fit(x, y)
    return my_model
```

```
my_model = train_your_model(x_train, y_train)
```

```
y_pred = my_model.predict(x_test)
print('Test-set accuarcy score: ', accuracy_score(y_test, y_pred))
```

```
Test-set accuarcy score:  0.9766666666666667
```

Accuracy	Your scores
acc > 0.975	20 points
0.95 < acc <= 0.975	15 points
0.9 < acc <= 0.95	10 points
acc < 0.9	0 points

Part. 2, Questions (30%):

1. Why does a decision tree have a tendency to overfit to the training set? Is it possible for a decision tree to reach a 100% accuracy in the training set? please explain. List and describe at least 3 strategies we can use to reduce the risk of overfitting of a decision tree.

If the stopping criteria of a decision tree is loose, decision tree is capable of making most of the input data to be leaf nodes, and this behavior is called overfitting since the model is too closely aligned to a limited set of data points.

If the model is allowed to be trained to its full strength, it will ended up creating classification node for each of the attributes and reach a 100% accuracy in the training set.

We can use pre-pruning, post-pruning, random forest to reduce the risk of overfitting of a decision tree.

① pre-pruning: early stopping the growth of the decision tree by setting the threshold of information gain.

② post-pruning: first allows decision trees to grow to their full depth, then starts removing the branches of the trees.

③ random forest: the base learners are only decision trees, and uses bagging along with column sampling to form a robust model.

2. This part consists of three True/False questions. Answer True/False for each question and briefly explain your answer.

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

True, follows from the updated equation, the weights of all misclassified points will be multiplied by $\exp[-\alpha_t y_i h_t(x_i)]$ before normalization.

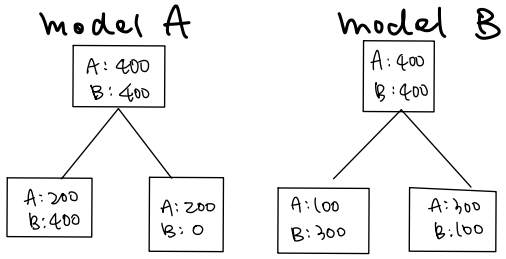
   b. In AdaBoost, weighted training error ε_t of the t_{th} weak classifier on training data with weights D_t tends to increase as a function of t.

True. 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 ε_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. If the data in the training set cannot be separated by a linear combination of the specific type of weak classifiers we are using, AdaBoost can't achieve zero training error.
For example consider the EXOR example with decision stumps as weak classifier. No matter how many iterations are performed, zero training error will not be achieved.

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.



model A misclassification rate : $\frac{200}{800} = 25\%$

model B misclassification rate : $\frac{100+100}{800} = 25\%$

$\Rightarrow$ both split produce a misclassification rate of 0.25

1. cross-entropy

• parent $= -\left[\frac{400}{800}\log_2\frac{400}{800} + \frac{400}{800}\log_2\frac{400}{800}\right] = 1$

① model A:

1) first node: $-\left(\frac{200}{600}\log_2\frac{200}{600} + \frac{400}{600}\log_2\frac{400}{600}\right)$

$= -\left(\frac{1}{3}\log_2\frac{1}{3} + \frac{2}{3}\log_2\frac{2}{3}\right)$

$\doteq 0.918$

2) second node: $-\left(\frac{200}{200}\left(\log_2\frac{200}{200} + 0\right)\right)$

$= 0$

$\Rightarrow$ total entropy $= \frac{600}{800} \times 0.918 + \frac{200}{800} \times 0$

$= \frac{3}{4} \times 0.918 \doteq 0.688$

$\Rightarrow$ gain $= 1 - 0.688 = 0.312$

② model B:

1) first node: $-\left(\frac{100}{400}\log_2\frac{100}{400}+\frac{300}{400}\log_2\frac{300}{400}\right)$

$= -\left(\frac{1}{4}\log_2\frac{1}{4}+\frac{3}{4}\log_2\frac{3}{4}\right)$

$\doteq 0.811$

2) second node: $-\left(\frac{300}{400}\log_2\frac{300}{400}+\frac{100}{400}\log_2\frac{100}{400}\right)$

$\doteq 0.811$

$\Rightarrow$ total entropy $= \frac{1}{2}\times 0.811+\frac{1}{2}\times 0.811 = 0.811$

$\Rightarrow$ gain $= 1-0.811 = 0.189$

2, Gini:

- parent: $1-\left(\frac{400}{800}\right)^2-\left(\frac{400}{800}\right)^2=\frac{1}{2}$

① model A:

1) first node: $1-\left(\frac{200}{600}^2+\frac{400}{600}^2\right)=1-\left(\frac{1}{3}\right)^2-\left(\frac{2}{3}\right)^2=\frac{4}{9}$

2) second node: $1-\left(\frac{200}{200}^2+0^2\right)=0$

$\Rightarrow$ total gini $= \frac{6}{8}\times\frac{4}{9}+\frac{2}{8}\times 0 \doteq 0.333$

$\Rightarrow$ gain $= \frac{1}{2}-0.333 \doteq 1.167$

② model B.

1) first node: $1-\left(\frac{100}{400}^2+\frac{300}{400}^2\right)=\frac{3}{8}$

2) second node: $1-\left(\frac{300}{400}^2+\frac{100}{400}^2\right)=\frac{3}{8}$

$\Rightarrow$ total gini $= \frac{1}{2}\times\frac{3}{8}+\frac{1}{2}\times\frac{3}{8}=\frac{3}{8}=0.375$

$\Rightarrow$ gain $= \frac{1}{2}-0.375=0.125$

$\Rightarrow$ Both the cross-entropy and gini index info gain is higher for the first split, and the first split is preferable due to purer node.