NYCU Pattern Recognition, Homework 3

0810749, 張君實

Part. 1, Coding (80%):

Logistic Regression Model

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.

```python
def gini(sequence):
    labels, counts = np.unique(sequence, return_counts=True)
    probs = counts/ float(np.shape(sequence)[0])
    gini = 1- np.sum(probs * probs)
    return gini


def entropy(sequence):
    labels, counts = np.unique(sequence, return_counts=True)
    probs = counts/ float(np.shape(sequence)[0])
    entropy = - np.sum(probs * np.log2(probs))
    return entropy
```

✓ 0.0s

```python
# 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 Python

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

```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)
#dt_depth3.traverse_tree(curr_node = dt_depth3.tree, mode = "print")
#dt_depth3.print_importance()
acc = accuracy_score(y_val, dt_depth3.predict(X_val))

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

✓ 0.6s Python

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

```python
""" Do Not Modify Below """

from sklearn.tree import DecisionTreeClassifier as SK_DecisionTreeClassifier

sk_dt = SK_DecisionTreeClassifier(criterion='gini', max_depth=3)
sk_dt.fit(X_train, y_train)
sk_acc = accuracy_score(y_val, sk_dt.predict(X_val))

assert round(acc, 3) == round(sk_acc, 3), "Because the Decision Tree without any trick has a fixed answer, you
```

✓ 0.1s Python

```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, sample_weight=None)

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

✓ 1.9s Python

```
Q2-2 max_depth=10:  0.8675
```

Q2-1 max_depth=3: 0.73125

Q2-2 max_depth=10: 0.8675

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)))
```

✓ 0.6s Python

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

```
# 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)))
```

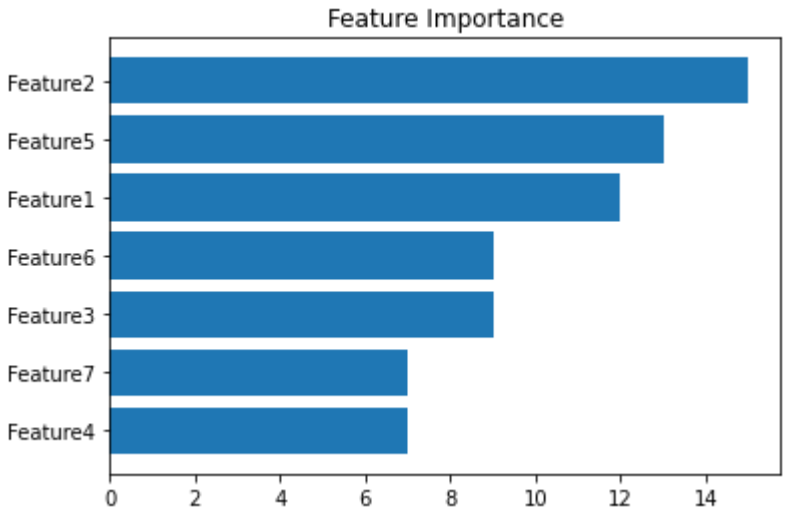
✓ 0.6s Python

```
Q3-2 criterion='entropy':  0.7725
```

criterion='gini': 0.73125

Q3-2 criterion='entropy': 0.7725

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', 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)))
```
✓ 4.6s Python

```
Q5-1 n_estimators=10:  0.89875
```

```
# 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("Q5-2 n_estimators=50: ", accuracy_score(y_val, rf_estimators50.predict(X_val)))
```
✓ 22.2s Python

```
Q5-2 n_estimators=50:  0.9
```

Q5-1 n_estimators=10: 0.89875

Q5-2 n_estimators=50: 0.9

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='gini', max_depth=None)
rf_maxfeature_sqrt.fit(X_train, y_train)

print("Q6-1 max_features='sqrt': ", accuracy_score(y_val,  rf_maxfeature_sqrt.predict(X_val)))
```
✓ 4.7s Python

```
Q6-1 max_features='sqrt':  0.89875
```

```
# 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("Q6-2 max_features='All': ", accuracy_score(y_val, rf_maxfeature_none.predict(X_val)))
```
✓ 13.8s Python

```
Q6-2 max_features='All':  0.875
```

Q6-1 max_features='sqrt': 0.89875

Q6-2 max_features='All': 0.875

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

First, I choose the random forest as my model, and I use the k-fold cross-validation and bootstrapping in this lab.

As for the k-fold cross-validation, I combine the training dataset and validation dataset, and I set the k=5, so I ultimately have 5 random forests model and 5 datasets, and these 5 forests will vote for prediction.

```python
# Build and train your model
class k_fold_forest():
    def __init__(self, k, n_estimators, max_features, boostrap, criterion, max_depth):
        self.k = k
        self.n_estimators = n_estimators
        self.max_features = max_features
        self.boostrap = boostrap
        self.criterion = criterion
        self.max_depth = max_depth
        self.forest_list = []

    def fit(self, X_train_k_fold, y_train_k_fold):
        average_acc = 0
        for i in range(self.k):
            X_train_ = np.concatenate((X_train_k_fold[0:np.int64(1600*i/self.k)], X_train_k_fold[np.int64(1600*(i+1)/self.k):1600]), axis=0)
            y_train_ = np.concatenate((y_train_k_fold[0:np.int64(1600*i/self.k)], y_train_k_fold[np.int64(1600*(i+1)/self.k):1600]), axis=0)
            X_val_ = X_train_k_fold[np.int64(1600*i/self.k) : np.int64(1600*(i+1)/self.k)]
            y_val_ = y_train_k_fold[np.int64(1600*i/self.k) : np.int64(1600*(i+1)/self.k)]
            your_model = RandomForest(n_estimators=self.n_estimators, max_features=self.max_features, boostrap=self.boostrap, criterion=self.criterion, max_depth=self.max_depth)
            your_model.fit(X_train_, y_train_)
            print(i," iteration:")
            print("The val of my model : ", accuracy_score(y_val_, your_model.predict(X_val_)))
            average_acc += accuracy_score(y_val_, your_model.predict(X_val_))
            self.forest_list.append(your_model)
        average_acc /= self.k
        print("average acc: ",average_acc)

    def predict(self, X_test_):
        num_data = np.shape(X_test_)[0]
        voting = np.zeros((self.k, num_data))
        prediction = np.zeros((num_data))

        for i,forest in enumerate(self.forest_list):
            voting[i] = forest.predict(X_test_)

        voting = np.transpose(voting, (1, 0))

        for i in range(num_data):
            values, counts = np.unique(voting[i], return_counts=True)
            prediction[i] = values[np.argmax(counts)]

        return prediction
```

```python
np.random.seed(0)
your_model = k_fold_forest(k = 5,n_estimators=450, max_features=2, boostrap=True, criterion='entropy', max_depth=8)
your_model.fit(X_train_k_fold, y_train_k_fold)
val_pred = your_model.predict(X_val)

acc = accuracy_score(y_val, val_pred)
print("total val acc: ", acc)
```

✓ 20m 33.0s Python

```
0  iteration:
The val of my model :  0.8875
1  iteration:
The val of my model :  0.934375
2  iteration:
The val of my model :  0.9
3  iteration:
The val of my model :  0.90625
4  iteration:
The val of my model :  0.90625
average acc:  0.906875
total val acc:  0.90625
```

According to my experience, I fine tune some hyperparameter of the random forests. I set t he

n_estimators=450,

max_features=2,

boostrap=True,

criterion='entropy',

max_depth=8

At last, the average accuracy of all forests is 90.685%, which is over 90.625%. Because I u se the k-fold cross-validation and bootstrapping, I think that the prediction results of the m odel are robust and reliable enough to reach the baseline.

Part. 2, Questions (20%):

1. Answer the following questions in detail:

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

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

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

a. When a decision tree is particularly deep, it tends to overfit the training set. First, the sample dataset is usually not big enough, and there are some noises in the dataset. Additionally, the unpruned decision tree could perfectly fit the training dataset. It classifies the nodes until it makes all leaves pure. In other words, it also takes the noise into consideration, but the noise is not helpful for predicting the data. Thus, the decision tree tends to overfit the training set.

b. Yes, it is possible for a decision tree to achieve 100% accuracy on the training set. For an unpruned decision tree, it would perfectly classify every training data. It is because all leaves are pure. But for an pruned decision tree, maybe some of the leaves are not pure, the decision tree may not achieve 100% accuracy on the training set.

c. (1) Pre-pruning the decision tree: We set the maximum depth of the decision tree. When the node of decision tree grows to a certain depth(3~8, it is the hyperparameter), we could stop it to grow. It could prevent the tree to overly classify the noise/ useless information.
(2) Post-pruning the decision tree: We train the decision tree model to grow to its full depth, then removes the tree branches to prevent the model from overfitting.
(3) Build the random forest: We could train a lot of the decision trees with different parts of the datasets(bagging) or different hyperparameter. Ultimately, we make the trees to vote the prediction. Due to the voting, they could decide the generalized predictions and reduce overfitting.

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.

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

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

(a) Yes, if the samples misclassified, the weights will update according to the following formula. Because the epsilon_t is a small number, the updated weights is bigger than the original weights.

(b) True. In the period of boosting iterations, the weak classifiers are forced to classify more difficult examples. The weights will increase for examples that are repeatedly misclassified by the weak classifiers. Therefore, the weighted training error epsilon_t of the t_th weak classifier on the training data tends to increase.

(c) False. If the data in the training set cannot be separated by a linear combination of the specific type of weak classifiers, AdaBoost will not eventually give zero training error. For example, consider the exclusive or example with decision stumps as weak classifiers. No matter how many iterations they are, zero training error will not be achieved.

3. Consider a data set comprising 400 data points from class $\boldsymbol{C_1}$ and 400 data points from class $\boldsymbol{C_2}$. 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 $\boldsymbol{C_1}$ and m points are assigned to $\boldsymbol{C_2}$. 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 $\boldsymbol{Entropy} = -\sum_{k=1}^{K} \boldsymbol{p_k log_2 p_k}$ and Gini index $\boldsymbol{Gini} = \boldsymbol{1} - \sum_{k=1}^{K} \boldsymbol{p_k^2}$ 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 rates:

model A: $\frac{200+400}{800} \cdot \frac{200}{200+400} + \frac{200}{800} \cdot \frac{0}{200}$

$= \frac{1}{4} = 25\%$

model B: $\frac{300+100}{800} \cdot \frac{100}{300+100} + \frac{100+300}{800} \cdot \frac{100}{100+300}$

$= \frac{1}{4} = 25\%$

The misclassification rates for the two trees are equal.

Tree A:

Entropy:

Node1: $-\frac{1}{3}\log_2\frac{1}{3} - \frac{2}{3}\log_2\frac{2}{3} \approx 0.9183$

Node2: $-1\log_2 1 - 0\log_2 0 = 0$

Gini:

Node1: $1 - \frac{1}{3}^2 - \frac{2}{3}^2 = 0.4444$

Node2: $1 - 1^2 - 0^2 = 0$

Tree B:

Entropy:

Node1: $-\frac{3}{4}\log_2\frac{3}{4} - \frac{1}{4}\log_2\frac{1}{4} = 0.8113$

Node2: $-\frac{1}{4}\log_2\frac{1}{4} - \frac{3}{4}\log_2\frac{3}{4} = 0.8113$

Gini:

Node1: $1 - \frac{3}{4}^2 - \frac{1}{4}^2 = \frac{3}{8} = 0.375$

Node2: $1 - \frac{1}{4}^2 - \frac{3}{4}^2 = \frac{3}{8} = 0.375$