Homework 2

COSC 3337 Dr.Rizk

In [1]:
```
%reload_ext watermark
%watermark -d -u -a '<Kim Nguyen>' -v -p numpy,scipy,matplotlib,sklearn
```

```
<Kim Nguyen>
last updated: 2020-06-18

CPython 3.7.6
IPython 7.12.0

numpy 1.18.1
scipy 1.4.1
matplotlib 3.1.3
sklearn 0.22.1
```

In [2]:
```
import numpy as np
```

1) Implementing an ID3 Decision Tree

1.1 Splitting a node

In [3]:
```
def split(array):
    # your code to generate dictionary
    res = {classi: np.where(array == classi)[0] for classi in np.unique(
array)}
    return res # return the dictionary variable
```

In [4]:
```
# Double check solution
print(split(np.array([0, 1, 2])))
print(split(np.array([1, 0, 1, 0, 0, 1, 0])))
print(split(np.array([1, 0, 3, 2, 0, 1, 1])))
```

```
{0: array([0]), 1: array([1]), 2: array([2])}
{0: array([1, 3, 4, 6]), 1: array([0, 2, 5])}
{0: array([1, 4]), 1: array([0, 5, 6]), 2: array([3]), 3: array([2])}
```

1.2 Implement Entropy

In [5]:
```
def entropy(array):
    # your code
    N = len(array)
    counts = np.bincount(array)
    P = counts[np.nonzero(counts)]/N # avoid log(0)
    H = -np.dot(P, np.log2(P))
    return H # return a scalar
```

In [6]:
```
# Double check solution
print(round(entropy(np.array([0, 1, 0, 1, 1, 0])), 4))
print(round(entropy(np.array([1, 2])), 4))
print(round(entropy(np.array([1, 1])), 4))
print(round(entropy(np.array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])), 4))
print(round(entropy(np.array([0, 0, 0])), 4))
print(round(entropy(np.array([1, 1, 1, 0, 1, 4, 4, 2, 1])), 4))
```

```
1.0
1.0
-0.0
0.4395
-0.0
1.6577
```

1.3 Implement Information Gain

In [7]:
```
def information_gain(x_array, y_array):
    parent_entropy = entropy(x_array)

    split_dict = split(y_array)

    for val in split_dict.values():
        freq = val.size/x_array.size
        child_entropy = entropy([x_array[i] for i in val])
        parent_entropy -= child_entropy*freq

    return parent_entropy
```

In [8]:
```
# Double check solution

x = np.array([0, 1, 0, 1, 0, 1])
y = np.array([0, 1, 0, 1, 1, 1])
print(round(information_gain(x, y), 4))

x = np.array([0, 0, 1, 1, 2, 2])
y = np.array([0, 1, 0, 1, 1, 1])
print(round(information_gain(x, y), 4))
```

```
0.4591
0.2516
```

1.4 Decision Tree Splitting

In [9]:
```
def make_tree(X, y):
# Return array if node is empty or pure (1 example in leaf node)

    if y.shape[0] == 1 or y.shape[0] == 0:
        return y

    # Compute information gain for each feature
    gains = [information_gain(x_attribute,y) for x_attribute in X.T] # Y
OUR CODE

    # Early stopping if there is no information gain
    if all(eachGain <= 1e-05 for eachGain in gains):
        return y # YOUR CODE

    # Else, get best feature
    best_feature = np.argmax(gains)

    results = {}

    # Use the `split` function to split on the best feature
    subset_dict = split(X[:, best_feature])

    # Note that each entry in the dictionary returned by
    # split is an attribute_value:array_indices pair.
    # here, we are going to iterate over these key-value
    # pairs and select the respective examples for the
    # new child nodes

    for feature_value, train_example_indices in subset_dict.items():
        child_y_subset = y.take(train_example_indices, axis = 0) # YOUR
 CODE
        child_x_subset = X.take(train_example_indices, axis = 0) # YOUR
 CODE

        # Next, we are using "recursion," that is, calling the same
        # tree_split function on the child subset(s)

        results["X_%d = %d" % (best_feature, feature_value)] = \
            make_tree(child_x_subset, child_y_subset)

    return results
```

In [10]:
```
# Double check solution
x1 = np.array([0, 0, 1, 1, 2, 2])
x2 = np.array([0, 1, 0, 1, 0, 1])
X = np.array([x1, x2]).T
y = np.array([0, 1, 0, 1, 1, 1])

print('Inputs:\n', X)
print('\nLabels:\n', y)

print('\nDecision tree:\n', make_tree(X, y))
```

```
Inputs:
 [[0 0]
 [0 1]
 [1 0]
 [1 1]
 [2 0]
 [2 1]]

Labels:
 [0 1 0 1 1 1]

Decision tree:
 {'X_1 = 0': {'X_0 = 0': array([0]), 'X_0 = 1': array([0]), 'X_0 = 2':
array([1])}, 'X_1 = 1': array([1, 1, 1])}
```

1.5 Building a Decision Tree API

In [11]:
```
class ID3DecisionTreeClassifer(object):

    def __init__(self):
        pass

    def fit(self, X, y):
        self.splits_ = make_tree(X,y) #YOUR CODE

    def _majority_vote(self, label_array):
        return np.argmax(np.bincount(label_array)) #YOUR CODE

    def _traverse(self, x, d):
        if isinstance(d, np.ndarray):
            return d
        for key in d:
            name, value = key.split(' = ')
            feature_idx = int(name.split('_')[-1])
            value = int(value)
            if x[feature_idx] == value:
                return self._traverse(x, d[key])

    def predict(self, x):

        label_array = self._traverse(x,self.splits_) #YOUR CODE to get c
lass labels from the target node
        return self._majority_vote(label_array) #YOUR CODE to predict th
e class label via majority voting from label_array
```

In [12]:
```
# Double check solution

tree = ID3DecisionTreeClassifer()
tree.fit(X, y)

print(tree.predict(np.array([0, 0])))
print(tree.predict(np.array([0, 1])))
print(tree.predict(np.array([1, 0])))
print(tree.predict(np.array([1, 0])))
print(tree.predict(np.array([1, 1])))
print(tree.predict(np.array([2, 0])))
print(tree.predict(np.array([2, 1])))
```

```
0
1
0
0
1
1
1
```

2) Bagging

2.1 Bootrapping

In [13]:
```
# DO NOT EDIT OR DELETE THIS CELL

from mlxtend.data import iris_data
X, y = iris_data()

print('Number of examples:', X.shape[0])
print('Number of features:', X.shape[1])
print('Unique class labels:', np.unique(y))
```

```
Number of examples: 150
Number of features: 4
Unique class labels: [0 1 2]
```

In [14]:
```
from sklearn.model_selection import train_test_split #YOUR CODE

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 45
, random_state=123, stratify=y) # YOUR CODE

print('Number of training examples:', X_train.shape[0])
print('Number of test examples:', X_test.shape[0])
```

```
Number of training examples: 105
Number of test examples: 45
```

In [15]:
```
# Double check solution
print('Number of training examples:', X_train.shape[0])
print('Number of test examples:', X_test.shape[0])
```

```
Number of training examples: 105
Number of test examples: 45
```

In [16]:
```
def draw_bootstrap_sample(rng, X, y):
    sample_indices = np.arange(X_train.shape[0]) # YOUR CODE
    bootstrap_indices = rng.choice(sample_indices,size=sample_indices.sh
ape[0]) # YOUR CODE
    return X[bootstrap_indices], y[bootstrap_indices] # YOUR CODE
```

In [17]:
```
# Double check solution

rng = np.random.RandomState(123)
X_boot, y_boot = draw_bootstrap_sample(rng, X_train, y_train)

print('Number of training inputs from bootstrap round:', X_boot.shape[0
])
print('Number of training labels from bootstrap round:', y_boot.shape[0
])
print('Labels:\n', y_boot)
```

```
Number of training inputs from bootstrap round: 105
Number of training labels from bootstrap round: 105
Labels:
 [0 0 1 0 0 1 2 0 2 1 0 0 2 1 1 1 1 2 1 1 2 0 2 1 2 1 1 1 0 1 0 0 1 2 0
0 0
 0 2 1 1 2 1 2 1 1 2 1 2 0 1 1 2 2 1 0 1 0 2 2 0 1 0 2 0 0 0 0 1 2 0 0
1 0
 1 1 0 1 1 2 2 0 2 0 2 0 1 1 2 2 0 2 2 2 0 1 0 1 2 2 2 1 0 0 0]
```

2.2 Bagging classifier from decision trees

In [18]:
```
from sklearn.tree import DecisionTreeClassifier

class BaggingClassifier(object):
    def __init__(self, num_trees=10, random_state=123):
        self.num_trees = num_trees
        self.rng = np.random.RandomState(random_state)

    def fit(self, X, y):
        self.trees_ = [DecisionTreeClassifier(random_state=self.rng) for
i in range(self.num_trees)]
        for i in range(self.num_trees):
            X_boot, y_boot = draw_bootstrap_sample(self.rng, X, y) #YOUR
CODE
            # YOUR CODE to
            # fit the trees in self.trees_ on the bootstrap samples
            self.trees_[i].fit(X_boot,y_boot) #YOUR CODE

    def predict(self, X):
        ary = np.zeros((X.shape[0], len(self.trees_)), dtype=np.int)
        for i in range(len(self.trees_)):
            ary[:, i] = self.trees_[i].predict(X)

        maj = np.apply_along_axis(lambda x:
                                  np.argmax(np.bincount(x)),
                                  axis=1,
                                  arr=ary)
        return maj
```

In [19]:
```
# Double check solution

model = BaggingClassifier()
model.fit(X_train, y_train)

predictions = model.predict(X_test)

print('Individual Tree Accuracies:')

for tree in model.trees_:
    predictions = tree.predict(X_test)
    print('%.1f%%' % ((predictions == y_test).sum() / X_test.shape[0] *
100))

print('\nBagging Test Accuracy: %.1f%%' % ((predictions == y_test).sum()
/ X_test.shape[0] * 100))
```

```
Individual Tree Accuracies:
88.9%
93.3%
97.8%
93.3%
93.3%
93.3%
91.1%
97.8%
97.8%
97.8%

Bagging Test Accuracy: 97.8%
```

3) Bias-Variance Decomposition

3.1 Bias_Variance decomposition of the 0-1 Loss for Decision Trees

In [20]:
```
# DO NOT EDIT OR DELETE THIS CELL

rng = np.random.RandomState(123)

num_bootstrap = 200

all_pred = np.zeros((num_bootstrap, y_test.shape[0]), dtype=np.int)

for i in range(num_bootstrap):
    X_boot, y_boot = draw_bootstrap_sample(rng, X_train, y_train)
    pred = DecisionTreeClassifier(random_state=66).fit(X_boot, y_boot).p
redict(X_test)
    all_pred[i] = pred

main_predictions = np.apply_along_axis(lambda x:
                                       np.argmax(np.bincount(x)),
                                       axis=0,
                                       arr=all_pred)
```

In [21]:
```
# YOUR CODE
avg_bias = round(np.sum(main_predictions != y_test) / y_test.size,4)
print("Average Bias: ", avg_bias)
```

```
Average Bias:  0.0222
```

In [22]:
```
# YOUR CODE
# you probably need multiple
# lines of code and a for-loop

var = np.zeros(pred.shape)

for pred in all_pred:
    var += (pred != main_predictions).astype(np.int)
var /= num_bootstrap

avg_var = round(var.sum()/y_test.shape[0],4)
print("Average variance: ", avg_var)
```

```
Average variance:  0.0346
```

3.2 Bias_Variance decomposition of the 0-1 Loss for Bagging

In [23]:
```
# YOUR SOLUTION
# Many lines of code (which you may copy and modify from 3.1)

rng = np.random.RandomState(123)
num_bootstrap = 200
all_pred = np.zeros((num_bootstrap, y_test.shape[0]), dtype=np.int)

for i in range(num_bootstrap):
    X_boot, y_boot = draw_bootstrap_sample(rng, X_train, y_train)
    model = BaggingClassifier()
    model.fit(X_boot, y_boot)
    pred= model.predict(X_test)
    all_pred[i] = pred

main_predictions = np.apply_along_axis(lambda x:
                                       np.argmax(np.bincount(x)),
                                       axis=0,
                                       arr=all_pred)
```

In [24]:
```
# YOUR CODE
avg_bias = round(np.sum(main_predictions != y_test) / y_test.size,4)
print("Average Bias: ", avg_bias)
```

```
Average Bias:  0.0222
```

In [25]:
```
# YOUR CODE
# you probably need multiple
# lines of code and a for-loop

var = np.zeros(pred.shape)

for pred in all_pred:
    var += (pred != main_predictions).astype(np.int)
var /= num_bootstrap

avg_var = round(var.sum()/y_test.shape[0],4)
print("Average variance: ", avg_var)
```

```
Average variance:  0.0281
```

Compare to the Bias_Variance decomposition of the 0-1 Loss for Decision Trees in 3.1

```
### => Average Bias is the same (0.0222)
### => Average variance is lower than in 3.1 (0.0281 < 0.0346)
```

3.3 Bias-Variance decomposition of the 0-1 Loss for AdaBoost

In [26]:
```
# YOUR SOLUTION
# Many lines of code (which you may copy and modify from 3.1)

from sklearn.ensemble import AdaBoostClassifier

rng = np.random.RandomState(123)
num_bootstrap = 200
all_pred = np.zeros((num_bootstrap, y_test.shape[0]), dtype=np.int)

for i in range(num_bootstrap):
    X_boot, y_boot = draw_bootstrap_sample(rng, X_train, y_train)
    pred = AdaBoostClassifier(random_state=66).fit(X_boot, y_boot).predi
ct(X_test)
    all_pred[i] = pred

main_predictions = np.apply_along_axis(lambda x:
                                       np.argmax(np.bincount(x)),
                                       axis=0,
                                       arr=all_pred)
```

In [27]:
```
# YOUR CODE
avg_bias = round(np.sum(main_predictions != y_test) / y_test.size,4)
print("Average Bias: ", avg_bias)
```

```
Average Bias:  0.0444
```

In [28]:
```
var = np.zeros(pred.shape)

for pred in all_pred:
    # (np.array(something) == np.array(something)).astype(np.int), the c
ondition is only for numpy array
    var += (pred != main_predictions).astype(np.int)
var /= num_bootstrap

avg_var = round(var.sum()/y_test.shape[0],4)

print("Average variance: ", avg_var)
```

```
Average variance:  0.0328
```

Compare to the Bias_Variance decomposition of the 0-1 Loss for Decision Trees in 3.1

```
### => Average Bias is higher than in 3.1 (0.0444 > 0.0222)
### => Average variance is a litlle lower than in 3.1 (0.0328 > 0.0346)
```

In [ ]: