Machine Learning
Homework #3
Question 4 & 5 Supplementary

Daniel Ho Kwan Leung
104360098
CSIE, Year 3

Question 4 & 5:

The original submitted question 4 was actually correct, but the following re-implementation is better and more general to cases. The section of question 5 is refined as the following implementation. The code enclosed is for both question 4 and 5.

For question 4, the data are encoded in integers as follow,

Attribute	Values	Encoded
outlook	sunny	1
Outlook	Overcast	2
Outlook	Rain	3
Temperature	≥ 80	3
Temperature	≥ 70 and ≤ 79	2
Temperature	< 70	1
Humidity	≥ 76	2
Humidity	< 76	1
Windy	True	1
Windy	False	0

and labels are encoded as follow,

play	1
no play	0

For question 5, the data are encoded in integers as follow,

Attribute	Values	Encoded
outlook	sunny	1
Outlook	Overcast	2
Outlook	Rain	3
Temperature	Determined by program	Determined by program
Temperature	Determined by program	Determined by program
Temperature	Determined by program	Determined by program
Humidity	Determined by program	Determined by program
Humidity	Determined by program	Determined by program
Windy	True	1
Windy	False	0

and labels are encoded as follow,

play	1
no play	0

The code link:

https://github.com/DHKLeung/NTUT_Machine_Learning/blob/master/HW3_Q4%26Q5.ipynb

```
import numpy as np
"""
Create data for decision tree
"""
def create_data(discrete=False):
    feature_name = np.array(['outlook', 'temperature', 'humidity', 'windy'])
    continuous_features_index = np.array([1, 2], dtype=np.float32) if not discrete
else np.empty((0), dtype=np.float32)
    data = np.array([
        [1., 85., 85., 0.],
        [1., 80., 90., 1.],
        [2., 83., 78., 0.],
        [3., 70., 96., 0.],
        [3., 68., 80., 0.],
        [3., 65., 70., 1.],
        [2., 64., 65., 1.],
        [1., 72., 95., 0.],
        [1., 69., 70., 0.],
        [3., 75., 80., 0.],
        [1., 75., 70., 1.],
        [2., 72., 90., 1.],
        [2., 81., 75., 0.],
        [3., 71., 80., 1.]
    ], dtype=np.float32)
    if discrete:
        data[:, 1][data[:, 1] < 70] = 1.
        data[:, 1][np.logical_and(70. <= data[:, 1],data[:, 1] <= 79.)] = 2.
        data[:, 1][data[:, 1] >= 80.] = 3.
        data[:, 2][data[:, 2] < 76] = 1.
        data[:, 2][data[:, 2] >= 76] = 2.
    label = np.array([
        [0.],
        [0.],
        [1.],
        [1.],
        [1.],
        [0.],
        [1.],
        [0.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [0.]
    ], dtype=np.float32)
    return data, label, feature_name, continuous_features_index
"""
Compute the entropy
"""
def get_entropy(x):
    distinct_x, count_x = np.unique(np.squeeze(x), return_counts=True)
    p = np.divide(count_x, np.sum(count_x))
    entropy = -np.sum(np.multiply(p, np.log2(p)))
    return entropy
"""
```

```
Select the best feature for splitting
Return: the index of the best features
"""
def get_best_feature(data, label, continuous_features_index):
    num_data = data.shape[0]
    num_feature = data.shape[1]
    base_entropy = get_entropy(label)
    information_gain_ratios = np.zeros((num_feature))
    best_low_mid_threshold = dict()
    best_mid_high_threshold = dict()
    best_data = dict()

    #Compute entropy for each feature given#
    for i in range(num_feature):
        #Dealing with continuous feature#
        if i in continuous_features_index:
            best_info_gain_ratio = 0.
            sorted_unique_values = np.sort(np.unique(data[:, i]))
            for j in range(sorted_unique_values.shape[0] - 1):
                for k in range(j + 1, sorted_unique_values.shape[0]):
                    data_temp = np.copy(data)
                    data_temp[:, i][data_temp[:, i] <= sorted_unique_values[j]] = 1.
                    data_temp[:, i][np.logical_and(sorted_unique_values[j] <
data_temp[:, i], data_temp[:, i] <= sorted_unique_values[k])] = 2.
                    data_temp[:, i][data_temp[:, i] > sorted_unique_values[k]] = 3.
                    unique_values = np.unique(data_temp[:, i])
                    feature_entropy = 0.
                    for value in unique_values:
                        label_per_value = label[data_temp[:, i] == value]
                        p_per_value = label_per_value.shape[0] / num_data
                        feature_entropy += p_per_value * get_entropy(label_per_value)
                    split = get_entropy(data_temp[:, i])
                    inform_gain_ratio = (base_entropy - feature_entropy) / split
                    if inform_gain_ratio > best_info_gain_ratio:
                        best_info_gain_ratio = inform_gain_ratio
                        best_low_mid_threshold[str(i)] = sorted_unique_values[j]
                        best_mid_high_threshold[str(i)] = sorted_unique_values[k]
                        best_data[str(i)] = np.copy(data_temp)

        #Dealing with discrete feature#
        else:
            unique_values = np.unique(data[:, i])
            feature_entropy = 0.
            for value in unique_values:
                label_per_value = label[data[:, i] == value]
                p_per_value = label_per_value.shape[0] / num_data
                feature_entropy += p_per_value * get_entropy(label_per_value)
            split = get_entropy(data[:, i])
            information_gain_ratios[i] = (base_entropy - feature_entropy) / split

    #Select the best feature to split the tree#
    best_feature_index = np.argmax(information_gain_ratios)

    #Return transformed data according to processed continuous feature#
    if best_feature_index in continuous_features_index:
        data_transformed = best_data[str(best_feature_index)]
        low_mid_threshold = best_low_mid_threshold[str(best_feature_index)]
        mid_high_threshold = best_mid_high_threshold[str(best_feature_index)]
    else:
        data_transformed = data
        low_mid_threshold = mid_high_threshold = 0.
    return best_feature_index, data_transformed, (low_mid_threshold,
mid_high_threshold)
"""
Create decision tree node
```

```
"""
def create_treenode(tree, data, label, feature_name, continuous_features_index):
    #Check if the data subset of the current node is pure#
    if np.unique(label).shape[0] == 1:
        return label.reshape(-1)[0]

    #Check if no features are remained#
    elif data.shape[1] == 0:
        possible_class, counts = np.unique(label, return_counts=True)
        return possible_class[np.argmax(counts)]

    #Get the best feature for splitting#
    best_index, data_transformed, contin_threshold = get_best_feature(data, label,
continuous_features_index)
    values = np.unique(data_transformed[:, best_index])
    split_feature_name = feature_name[best_index]

    #Update the index of continuous feature and the feature name list#
    new_continuous_features_index = np.copy(continuous_features_index)
    if best_index in new_continuous_features_index:
        new_continuous_features_index = np.setdiff1d(new_continuous_features_index,
np.array([best_index], dtype=np.float32))
    new_continuous_features_index[best_index <= new_continuous_features_index] -= 1
    new_feature_name = np.append(feature_name[0:best_index], feature_name[best_index
+ 1:])

    #Create the tree#
    for value in values:
        #Create new data and label for child node#
        temp = data_transformed[data_transformed[:, best_index] == value]
        new_data = np.append(temp[:, :best_index], temp[:, best_index + 1:], axis=1)
        new_label = label[data_transformed[:, best_index] == value]

        #Recursion#
        subtree = create_treenode(list(), new_data, new_label, new_feature_name,
new_continuous_features_index)

        #Dealing with continuous feature#
        if best_index in continuous_features_index:
            if value == 1.:
                cond_str = '<= {}'.format(contin_threshold[0])
            elif value == 2.:
                cond_str = '{} <  & <= {}'.format(contin_threshold[0],
contin_threshold[1])
            else:
                cond_str = '{} <'.format(contin_threshold[1])
            tree.append(((split_feature_name, cond_str), subtree))
        else:
            tree.append(((split_feature_name, value), subtree))
    return tree
```

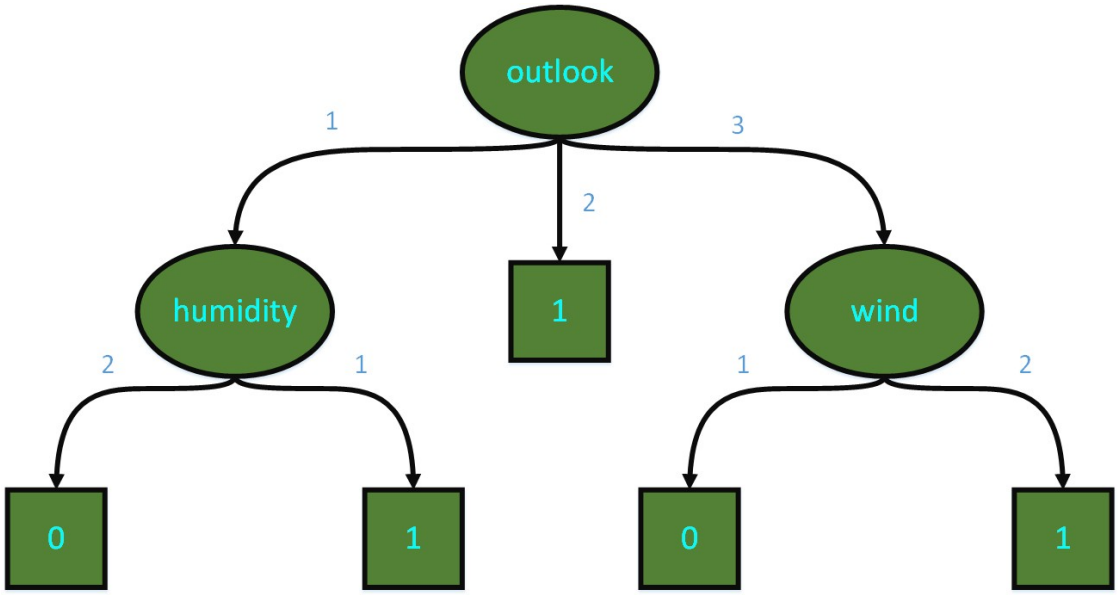
```
data, label, feature_name, continuous_features_index = create_data(discrete=True)
tree = create_treenode(list(), data, label, feature_name, continuous_features_index)
tree
```

```
data, label, feature_name, continuous_features_index = create_data(discrete=False)
tree = create_treenode(list(), data, label, feature_name, continuous_features_index)
tree
```

For question 4, the result is

```
[(('outlook', 1.0), [(('humidity', 1.0), 1.0), (('humidity', 2.0), 0.0)]),
 (('outlook', 2.0), 1.0),
 (('outlook', 3.0), [(('windy', 0.0), 1.0), (('windy', 1.0), 0.0)])]
```

which denotes



For question 5, the result is

```
[(('outlook', 1.0),
  [(('windy', 0.0),
    [(('temperature', '<= 69.0'), 1.0),
     (('temperature', '69.0 < & <= 85.0'), 0.0)]),
   (('windy', 1.0),
    [(('temperature', '<= 75.0'), 1.0),
     (('temperature', '75.0 < & <= 80.0'), 0.0)])]),
 (('outlook', 2.0), 1.0),
 (('outlook', 3.0), [(('windy', 0.0), 1.0), (('windy', 1.0), 0.0)])]
```

The way to search for best division for continuous type of feature is using exhaustive search. In this question, we are required to search for three ranges for temperature and humidity.

The algorithm should be like

Exhaustive Search for three ranges in question 5

Input: $\mathbf{D}$, the list of continuous-type values
$\mathbf{L}$, the label of the corresponding order of $\mathbf{D}$
Output: $\mathbf{Threshold_{LM}}$, the threshold between "low" and "mid"
$\mathbf{Threshold_{MH}}$, the threshold between "mid" and "high"

1. Obtain $\mathbf{D_{distinct}}$ by getting distinct values from $\mathbf{D}$ and sort by ascending order
2. **foreach** $\mathbf{V_{LM}}$ **in** $\mathbf{D_{distinct}}$
3. | **foreach** $\mathbf{V_{MH}}$ **in** $\mathbf{D_{distinct}}$ **where** $\mathbf{V_{LM} < V_{MH}}$
4. | | Copy $\mathbf{D_{copy}}$ from $\mathbf{D}$
5. | | Set all the values in $\mathbf{D_{copy}} \leq \mathbf{V_{LM}}$ be "low"
6. | | Set all the values in $\mathbf{D_{copy}} > \mathbf{V_{LM}} \,\&\, \leq \mathbf{V_{MH}}$ be "mid"
7. | | Set all the values in $\mathbf{D_{copy}} > \mathbf{V_{MH}}$ be "high"
8. | | Compute the entropy, information gain ratio for $\mathbf{D_{copy}}$

9.	Save the $\mathbf{V_{LM}}$, $\mathbf{V_{MH}}$ if the information gain ratio is the largest

The feature “humidity” is not used in this tree. “temperature” is used twice under different value of “windy”. Under “windy” = False, “low” of temperature is ≤ 69, “mid” of temperature is $69 < \& \leq 85$, “high” of temperature is > 85. Under “windy” = True, “low” of temperature is ≤ 75, “mid” of temperature is $75 < \& \leq 80$, “high” of temperature is > 80.

However, by the using of dataset given, “high” in temperature is not used in the tree generated since the sub-dataset brought to the temperature node doesn’t actually contain value in the range of “high”.

The generated decision tree is,