Name: Zongdi Xu
USC ID: 5900-5757-70

Assignment 1

Decision Trees

INF 552
February 3, 2019

Part 1: Implementation

For this assignment, I write my program in Python 2.7. It is easy to run on any platform installed wit compatible Python Interpreter.

1. Calculating the entropy of the given datasets. There are two types of entropy to be calculated when building a decision tree[1].

   a) Entropy based on the frequency of one attribute:

   $$E(X) = \sum -p_i log_2(p_i),$$

   while p_i refers to the frequency of every possible value of the attribute X.

   b) Entropy based on the frequency of the observed attribute and the target attribute:

   $$E(X, T) = \sum p_i E(T),$$

   while p_i refers to the frequency of every possible value of the observed attribute X, and $E(T)$ refers to the entropy of target attribute T within the very subset divided the i-th value of X. It can also be viewed as the weighed average of every entropy of subsets.

   Here is the psuedo code implementation of entropy calculation in Python-style. The function 'entropy' will automatically figure out the frequency of every value of the attribute, with the values of the input 'dataset' given. When given the parameter 'observe' which is equal to 'target', it will work on the type a), otherwise turn to the type b).

   Algorithm 1: Calculating the two types of entropy

```python
def entropy(dataset, list, observe, target):
    attr_cnt = {}
    sum = 0
    for id in list:
        attr_cnt[dataset[id][observe]].add(id)
        sum += 1

    e = 0.0

    for value in attr_cnt.itervalues():
        # value is a list of subset
        frequency = 1.0*len(value)/sum
        if observe == target: # type a)
            e += frequency*log(1.0/frequency)/log(2.0)
        else: # type b)
            e += frequency*entropy(dataset, value, target, target)
    return e
```

2. Build decision tree recursively. At first, specify a root of the tree attached to the whole input dataset. Try to divide the dataset into subsets and attached to the child node of the tree. The changes of the entropy before and after the observation of one attribute on current dataset is also defined as Information Gain:

$$Gain(X,T) = E(T) - E(X,T).$$

Algorithm 2: Building a decision tree

```
def recursive_build(dataset, node, attribute):

    target_entropy = entropy(
        dataset, node['__list__'], target, target)

    chosen = -1
    max_inf_gain = 0.0
    for observe in range(attribute['num']):
        if not attribute['isChosen'][observe]:
            branch_entropy = entropy(
                dataset, node['__list__'], observe, target)
            inf_gain = target_entropy - branch_entropy

            if inf_gain > max_inf_gain:
                max_inf_gain = inf_gain
                chosen = observe
    # find the observed attribute with the maximum information gain

    if chosen > -1:
        attribute['isChosen'][chosen] = True
        # prevent duplicate observation of one attribute
        node['__label__'] = attribute['caption'][chosen]
        node[dataset[id][chosen]]['__list__'] <- node['__list__']
        recursive_build(dataset.childs(), value, attribute)
        attribute['isChosen'][chosen] = False
    else:
        node is labeled as a leaf node
```

3. Print the every node of the decision tree.

Algorithm 3: Printing a decision tree

```
def recursive_print(node, level=0):
    if node.has_key('__label__'):
        print node['__label__']
        # the next attribute

    for key in node.keys():
        if not key == '__list__' and not key == '__label__':
            if node.has_key('__res__'): # this is a leaf node
                print node['__res__']
```

```
        else: # this node has child(s)
            print '␣␣'*(level+1)+key+':',
            recursive_print(node[key], level+1)
```

4. Read input data from file. Regular expression functions ("import re") are used to format the data from the input file (dt-data.txt).

Algorithm 4: Reading input file

```
def read_data(filename):
    input_f = open(filename, 'r')
    data = input_f.readlines()

    predictor = [re.sub(r'^\s+|\s+$', '', word)
                 for word in list(filter(None,
                 re.split(r'\(|\)|,|\n', data[0])))]
    # the names of every attribute

    attribute = {'num': len(predictor), 'caption': predictor,
    'isChosen': [
        False]*len(predictor)}

    tree = {'__list__': range(len(data)-2)}
    # initializing the root node of the tree

    dataset = [list(filter(None,
    re.split('␣|:|,|;|\n', rec)))[1:]
            for rec in data[2:]]
    # get every value of the dataset

    return dataset, tree, attribute

dataset, tree, attribute = read_data('dt-data.txt')
```

5. Test the printing function. See Table 1.

Table 1: The printing of the tree.

Occupied		
	High: Location	
	Talpiot: No	
	City-Center: Yes	
	German-Colony: No	
	Mahane-Yehuda: Yes	
Moderate: Location		
	Talpiot: Price	
		Normal: Yes
		Cheap: No
	City-Center: Yes	
	Ein-Karem: Yes	
	German-Colony: VIP	
		Yes: Yes
		No: No
	Mahane-Yehuda: Yes	
Low: Location		
	Talpiot: No	
	City-Center: Price	
		Normal: No
		Cheap: No
	Ein-Karem: Price	
		Normal: No
		Cheap: Yes
	Mahane-Yehuda: No	

6. Make a prediction on another given test data. The function will start at the root node and traverse to its childs to get the correct answer. See Algorithm 4 and Table 2.

Algorithm 5: Prediction

```python
def recursive_predict(testdata, dataset, node):
    if node.has_key('__res__'):
        return node['__res__']

    try:
        next = testdata.get(node['__label__'])
        return recursive_predict(testdata, dataset, node[next])
    except:
        return '[ERROR]'
test = {'Occupied': 'Moderate', 'Price': 'Cheap', 'Music': 'Loud',
'Location': 'City-Center', 'VIP': 'No', 'Favorite Beer': 'No'}

recursive_build(dataset, tree, attribute)
print 'The prediction test result of', test, 'is:',
recursive_predict(
    test, dataset, tree)
```

Table 2: The result of prediction.

The prediction test result of {'Price': 'Cheap', 'VIP': 'No', 'Music': 'Loud', 'Location': 'City-Center', 'Occupied': 'Moderate', 'Favorite Beer': 'No'} is: Yes

```
CitruXonves-MBP:hw1 crxon$ python decision_tree.py
Occupied
  High: Location
    Talpiot: No
    City-Center: Yes
    German-Colony: No
    Mahane-Yehuda: Yes
  Moderate: Location
    Talpiot: Price
      Normal: Yes
      Cheap: No
    City-Center: Yes
    Ein-Karem: Yes
    German-Colony: VIP
      Yes: Yes
      No: No
    Mahane-Yehuda: Yes
  Low: Location
    Talpiot: No
    City-Center: Price
      Normal: No
      Cheap: No
    Ein-Karem: Price
      Normal: No
      Cheap: Yes
    Mahane-Yehuda: No
The prediction test result of {'Price': 'Cheap', 'VIP': 'No', 'Music': 'Loud', 'Location': 'City-Center', 'Occupied': 'Moderate', 'Favorite Beer': 'No'} is: Yes
```

Figure 1: Screenshot of the printing and predicting result.

Part 2: Software Familiarization

'Scikit-learn' is a well-known library with pre-compiled classifiers from machine learning[2]. Compared to my implementation, it does not only support classification, but is designed for the needs regression[3]. So it appears to be a solution to broader problems. See Figure below for result.

Algorithm 6: Decision tree classifier in 'sklearn' library.

```
def main():
    data_x, data_y, enc_x, enc_y = read_data('dt-data.txt')
    test_x = [['Moderate', 'Cheap', 'Loud', 'City-Center', 'No', 'No']]

    dtc = tree.DecisionTreeClassifier()
    dtc = dtc.fit(enc_x.transform(data_x).toarray(),
                  enc_y.transform(data_y).toarray())

    print 'The prediction test result of', test_x, 'is:',
    enc_y.inverse_transform(dtc.predict(enc_x.transform(test_x).toarray()))
```

Yet it would make mistakes when I attempt to attach every categorial value to a certain numerical value. The mechanism of handling of classification problems is still different from that of regression problems[3]. Thus 'One Hot Encoding' is proposed to solve this problem[4].

Algorithm 7: One Hot Encoding.

```
def read_data(filename):
    input_f = open(filename, 'r')
    lines = input_f.readlines()

    predictor = [re.sub(r'^\s+|\s+$', '', word)
                 for word in list(filter(None,
                 re.split(r'\(|\)|,|\n', lines[0])))]

    data = [list(filter(None, re.split(' |:|,|;|\n', rec)))[1:]
            for rec in lines[2:]]

    data_x = [line[:-1] for line in data]
    data_y = [line[-1:] for line in data]
    enc_x = preprocessing.OneHotEncoder()
    enc_x.fit(data_x)
    enc_y = preprocessing.OneHotEncoder()
    enc_y.fit(data_y)

    return data_x, data_y, enc_x, enc_y
```

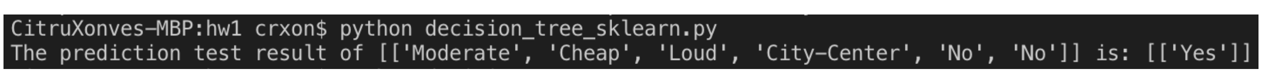

Figure 2: Screenshot of the printing and predicting result using 'sklearn'.

Part 3: Applications

Decision trees are highly preferred in the industry because of its simplicity to navigate and simulate what-if scenarios. This feature of decision trees was a apparent reason to gain popularity across many industries from insurance to retail sectors. It plays a role in credit risk scoring in the banking and financial services[5].

Actually, I have encountered such cases when credits available on my credit card are determined by the machine learning system based on my daily consuming behaviors.

References

[1] Decision Tree - Classification, https://www.saedsayad.com/decision_tree.htm.

[2] sklearn.tree.DecisionTreeClassifier — scikit-learn 0.20.2 documentation, https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

[3] Passing categorical data to Sklearn Decision Tree, https://stackoverflow.com/questions/38108832/passing-categorical-data-to-sklearn-decision-tree.

[4] sklearn.preprocessing.OneHotEncoder — scikit-learn 0.20.2 documentation, https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html.

[5] What are some practical business uses of decision trees? https://www.quora.com/What-are-some-practical-business-uses-of-decision-trees