Report for hw2

> 2018000337 장호우

Environment

- Windows 10
- Python 3.7.7 64-bit

Requirements

- numpy
- pandas

To install requirements

```
pip3 install -r requirements.txt
```

Content

Summary of your algorithm

To create a training model to predict the class or value of the target variable by learning simple decision rules inferred from training data. To train a classifier model from train data which has both attributes and label. Then classify a test data which only contains attributes to each label based on the trained algorithm. The train of decision tree algorithm starts with finding the best attribute. The best attribute means it draws the best information gain among given attributes. Then it splits the training set into left and right node with respect to the best attribute. It repeats this single process recursively until the conditions are achieved.

Detailed description of each function

```
# parse the argument
parser = argparse.ArgumentParser()
parser.add_argument("train", help="train data file", default='dt_train1.txt',
type=str)
parser.add_argument("test", help="test data file", default='dt_test1.txt',
type=str)
parser.add_argument("output", help="output path", default='dt_result1.txt',
type=str)
args = parser.parse_args()
```

```
# using pandas to load data from files
def load_data(train=args.train, test=args.test):
    train = pd.DataFrame(pd.read_csv(train, sep='\t'))
    test = pd.DataFrame(pd.read_csv(test, sep='\t'))
    return np.array(train).tolist(), np.array(test).tolist()
```

```
# the node structure of tree
class Node:
    def __init__(self, attr=None, value=None, leaf=None, left=None, right=None):
        self.attr = attr
        self.value = value
        self.leaf = leaf
        self.left = left
        self.right = right
```

```
#
def tree(dataset):
    # get labels of dataset
    def labels(dataset):
        label_counts = dict()
        for feat in dataset:
            label = feat[-1]
            if label not in set(label_counts): label_counts[label] = 0
            label_counts[label] += 1
        return label_counts
    # calculate the entropy
    def entropy(dataset):
        e = .0
        label_counts = labels(dataset)
        for key in label_counts:
            p = float(label_counts[key]) / len(dataset)
            e -= p * log(p, 2)
        return e

    base_e = entropy(dataset)
    best_gain, best_attr, best_splited = .0, None, None
    # loop every attributes
    for attr in range(0, len(dataset[0])-1):
        for v in set([row[attr] for row in dataset]):
            left, right = list(), list()
            # to split
            for row in dataset:
                left.append(row) if row[attr] == v else right.append(row)
            # calculate the information gain
            p = len(left) / float(len(dataset))
            gain = base_e - p*entropy(left) - (1-p)*entropy(right)
            # update the best information gain
            if gain > best_gain:
                best_gain, best_attr, best_splited = gain, (attr, v), (left,
right)
    # repeat the process until the information gain get to zero
    if best_gain > 0:
        return Node(attr=best_attr[0], value=best_attr[1],
left=tree(best_splited[0]), right=tree(best_splited[1]))
    else:
        return Node(leaf=labels(dataset))
```

```
# classify the test data by classified decision tree
def Classifier(data, tree):
    def classify(row, tree):
        if tree.leaf:
            return tree.leaf
        else:
            branch = tree.left if row[tree.attr] == tree.value else tree.right
            # task recursively
            return classify(row, branch)

    [data[i].append(list(classify(data[i], tree).keys())[0]) for i in
range(len(data))]
    return data
```

```
# main program
if __name__ == "__main__":
    # load the data
    train, test = load_data()
    # get header from training data but not necessary
    header = pd.DataFrame(pd.read_csv(args.train, sep='\t')).columns.values
    # save to the result.txt with header
    pd.DataFrame(Classifier(test, tree(train))).to_csv(args.output,
header=header, index=False, sep='\t')
```

How to run it

Directory Structure

```
└─assignment2
        2020_DM_Programming_Assignment_2.pdf
        dt.py
        dt_answer.txt
        dt_answer1.txt
        dt_result.txt
        dt_result1.txt
        dt_test.exe
        dt_test.txt
        dt_test1.txt
        dt_train.txt
        dt_train1.txt
        README.md
        report2.pdf
        requirements.txt
```

Running

For Example 1:

```
python dt.py dt_train.txt dt_test.txt dt_result.txt
```

For Example 2:

```
python dt.py dt_train1.txt dt_test1.txt dt_result1.txt
```

Testing

For Example 1:

```
./dt_test.exe dt_answer.txt dt_result.txt
```

output:

```
5 / 5
```

For Example 2:

```
./dt_test.exe dt_answer1.txt dt_result1.txt
```

output:

```
336 / 346
```