Homework 2

May 14, 2020

```
In [1]: import pandas as pd
        import numpy as np

In [2]: data_path = "./data/house-votes-84.data"

In [3]: data_df =  pd.read_table(data_path, header=None, sep=',')
        print("data_df.shape =",data_df.shape)

data_df.shape = (435, 17)


In [4]: # process raw data
        data_ar = np.array(data_df)
        n_att_raw = data_ar.shape[1]

        n_item = data_ar.shape[0]
        n_att = n_att_raw * 2

        data = np.zeros((n_item, n_att))
        for i in range(n_item):
            if data_ar[i][0] == 'democrat':
                data[i][0] = 1
            else:
                data[i][n_att_raw] = 1
            for j in range(1, n_att_raw):
                if data_ar[i][j] == 'y':
                    data[i][j] = 1
                elif data_ar[i][j] == 'n':
                    data[i][j + n_att_raw] = 1

In [5]: attribute_information = [
            'Class Name: democrat',
            'handicapped-infants: yes',
            'water-project-cost-sharing: yes',
            'adoption-of-the-budget-resolution: yes',
            'physician-fee-freeze: yes',
            'el-salvador-aid: yes',
            'religious-groups-in-schools: yes',
            'anti-satellite-test-ban: yes',
            'aid-to-nicaraguan-contras: yes',
            'mx-missile: yes',
            'immigration: yes',
            'synfuels-corporation-cutback: yes',
            'education-spending: yes',
            'superfund-right-to-sue: yes',
            'crime: yes',
            'duty-free-exports: yes',
            'export-administration-act-south-africa: yes',
            'Class Name: republican',
            'handicapped-infants: no',
            'water-project-cost-sharing: no',
            'adoption-of-the-budget-resolution: no',
            'physician-fee-freeze: no',
            'el-salvador-aid: no',
            'religious-groups-in-schools: no',
            'anti-satellite-test-ban: no',
            'aid-to-nicaraguan-contras: no',
            'mx-missile: no',
            'immigration: no',
            'synfuels-corporation-cutback: no',
            'education-spending: no',
            'superfund-right-to-sue: no',
            'crime: no',
            'duty-free-exports: no',
            'export-administration-act-south-africa: no'
        ]
        print(data)
```

```
[[0. 0. 1. ... 0. 1. 0.]
 [0. 0. 1. ... 0. 1. 0.]
 [1. 0. 1. ... 0. 1. 1.]
 ...
 [0. 0. 0. ... 0. 1. 0.]
 [0. 0. 0. ... 0. 1. 0.]
 [0. 0. 1. ... 0. 0. 1.]]
```

```
In [6]: def gen_next_fi(l):
            # generate candidate itemset according to last frequent itemset
            # l is a list containing all k-item frequent itemset
            # e.g. l=[{1,2,3},{2,3,5},{4,6,9},...]
            if not l:
                return []
            new_l = []
            k = len(l[0])
            len_l = len(l)
```

```
        for i in range(len_l):
            for j in range(i + 1, len_l):
                itemset1 = l[i]
                itemset2 = l[j]
                set1_test = sorted(itemset1)
                set2_test = sorted(itemset2)
                if set1_test[: k-1] == set2_test[: k-1]:
                    new_l.append(itemset1 | itemset2)
        return new_l
```

In [7]:
```
def find_k_fi(l, support):
    # generate frequent itemset according to the candidate itemset
    # e.g. l=[{1,2,3},{2,3,5},{4,6,9},...]
    if not l:
        return []
    global data
    n = len(data)
    d = {}
    k = len(l[0])

    # gerate keys for d
    for itemset in l:
        d[tuple(sorted(itemset))] = 0
        d[tuple(sorted(itemset))] = 0

    # compute support for each itemset
    for key in d.keys():
        for line in data:
            s = 0
            for a in key:
                if line[a]:
                    s += 1
            if s == k:
                d[key] += 1
    l_l = []
    for key in list(d.keys()):
        if d[key] / n < 0.3:
            del d[key]
        else:
            if set(key) not in l_l:
                l_l.append(set(key))

    return d, l_l
```

In [8]:
```
# init l_att_comb
l_att_comb = []
for i in range(n_att):
    l_att_comb.append({i})
```

```
        # l_att_comb is a list containing all 1-item frequent itemset
        # gen_next_fi(l_att_comb)is a list containing all 2-item frequent itemset
        # etc

        # F[i] contain frequent itemset and support with the length of itemset is i+1
        F = []

        for k in range(n_att):
            print('k:', k)
            print('length of l_k):', len(l_att_comb))
            c_k, l_next = find_k_fi(l_att_comb, support=0.3)
            F.append(c_k)
            l_att_comb = gen_next_fi(l_next)
            if not len(l_att_comb):
                break
        print("k_max =", k)
        print("done")
```

```
k: 0
length of l_k): 34
k: 1
length of l_k): 528
k: 2
length of l_k): 693
k: 3
length of l_k): 700
k: 4
length of l_k): 332
k: 5
length of l_k): 123
k: 6
length of l_k): 32
k: 7
length of l_k): 3
k_max = 7
done
```

```
In [9]: def get_subset(s):
            # return all subsets of set s
            # s is iterable and return is a list
            n = len(s)
            result = []
            for i in range(2 ** n):
                combo = []
                for j in range(n):
                    if(i >> j) % 2:
                        combo.append(s[j])
                result.append(combo)
            result.remove([])
            result.remove(list(s))

            return result
```

```
In [10]: def gen_rules(k_, confidence):
             global F, data
             r = []
             f = F[k_-1]
             for itemset, sup in f.items():
                 # e.g. itemset==(0, 3, 8, 16)
                 #      sup==138
                 sup1 = sup
                 subset = get_subset(itemset)
                 for ss in subset:
                     # ss is a sub set of itemset
                     # conf : sup(itemset) / sup(ss)
                     # rule : ss -> itemset - ss
                     if len(ss) > 1:
                         sup2 = F[len(ss)-1][tuple(sorted(ss))]
                         conf = sup1 / sup2
                         if conf >= confidence:
                             r.append((set(ss), set(itemset)-set(ss), conf))
             return r
```

```
In [11]: rules = [[],]
         # rules[i] contain (rule, support) with the length of itemset is i
         for i in range(1, k+1):
             rules.append(gen_rules(i, confidence=0.9))
```

```
In [12]: def save_result(result, path):
             global attribute_information
             with open(path, 'w') as f:
                 for rs_k in result:
                     if rs_k:
                         for a, b, conf in rs_k:
                             # e.g. a, b, conf =({7, 9}, {0}, 0.971)
                             a = set([attribute_information[i] for i in a])
                             b = set([attribute_information[i] for i in b])
                             conf = "confidence: {}".format(conf)
                             f.write(str((a, b, conf)) + '\n')
```

```
In [13]: save_path = './result.txt'
         save_result(rules, save_path)
```

```
In [ ]:
```