HW2

October 29, 2018

```
In [1]: import numpy as np
        import urllib
        import networkx as nx
        import matplotlib.pyplot as plt
        from homework2_starter import *

        %load_ext autoreload
        %autoreload 2
```

1 Classifier evaluation

```
In [2]: data = list(parseData("http://jmcauley.ucsd.edu/cse190/data/beer/beer_50000.json"))
        X = [feature(d) for d in data]
        y = [d['beer/ABV'] >= 6.5 for d in data]

        X = np.array(X)
        y = np.array(y).reshape(-1, 1)
```

```
In [3]: ##################################################
        # 1                                              #
        ##################################################

        def split_data(X, Y, val_ratio, test_ratio, shuffle=False):
            m = X.shape[0]

            m_train = int(m * (1 - val_ratio - test_ratio))
            m_val = int(m * val_ratio) + m_train
            m_test = int(m * test_ratio) + m_val

            if shuffle:
                permutation = np.random.permutation(m)
                X = X[permutation, :]
                Y = Y[permutation, :]

            return (X[:m_train, :], Y[:m_train, :], X[m_train:m_val, :],
                    Y[m_train:m_val, :], X[m_val:m_test, :], Y[m_val:m_test, :])
```

```
def evaluation(theta, X, y):
    scores = [inner(theta,x) for x in X]
    predictions = [s > 0 for s in scores]
    correct = [(a==b) for (a,b) in zip(predictions,y)]
    acc = sum(correct) * 1.0 / len(correct)
    return acc

X_train, y_train, X_val, y_val, X_test, y_test = split_data(X, y, 1/3, 1/3, True)

lam = 1.
theta = train(lam, X, X_train, y_train)
acc = evaluation(theta, X_val, y_val)
print("Validation set accuracy=%f" % acc)
acc = evaluation(theta, X_test, y_test)
print("Test set accuracy=%f" % acc)
```

```
Validation set accuracy=0.720449
Test set accuracy=0.716789
```

In [4]:
```
##################################################
# 2                                              #
##################################################
def confusion_mat(theta, X, y):
    scores = [inner(theta,x) for x in X]
    predictions = [s > 0 for s in scores]

    tp = 0
    fp = 0
    fn = 0
    tn = 0

    for (a, b) in zip(predictions, y):
        if a == b:
            if a == True: tp += 1
            else: tn += 1
        else:
            if a == True: fp += 1
            else: fn += 1
    return tp, fp, fn, tn

tp, fp, fn, tn = confusion_mat(theta, X_test, y_test)
print('Positive:%d\tNegative:%d\nTP:%d\tTN:%d\tFP:%d\t:FN:%d\t' %
      (tp+fp, fn+tn, tp, tn, fp, fn))
```

```
Positive:12300        Negative:4366
TP:8943        TN:3003        FP:3357        :FN:1363
```

In [5]:
```
##################################################
# 3                                              #
##################################################
```

In [6]:
```
##################################################
# 4                                              #
##################################################
lambdas = [0, 0.01, 0.1, 1, 100]
best_lam = None
best_acc = {}

for lam in lambdas:
    theta = train(lam, X, X_train, y_train)
    val_acc = evaluation(theta, X_val, y_val)
    if best_lam is None or best_acc['val'] < val_acc:
        best_lam = lam
        best_acc['train'] = evaluation(theta, X_train, y_train)
        best_acc['val'] = val_acc
        best_acc['test'] = evaluation(theta, X_test, y_test)

print('Best lambda = %f' % best_lam)
print('Train acc=%f \tValidation acc=%f \tTest acc=%f' %
      (best_acc['train'], best_acc['val'], best_acc['test']))
```

```
Best lambda = 1.000000
Train acc=0.719969        Validation acc=0.720449        Test acc=0.716789
```

2 Community Detection

In [7]:
```
edges = set()
nodes = set()
for edge in urllib.request.urlopen("http://jmcauley.ucsd.edu/cse255/data/facebook/egonet
    x,y = edge.split()
    x,y = int(x),int(y)
    edges.add((x,y))
    edges.add((y,x))
    nodes.add(x)
    nodes.add(y)
```

In [8]:
```
##################################################
# 5                                              #
##################################################

def dfs(node):
    if node in visited:
        return
    comp.add(node)
```

```python
            visited.add(node)
            for edge in edges:
                if node == edge[0]:
                    dfs(edge[1])

        visited = set()
        comps = []
        comps_len = []

        for node in nodes:
            comp = set()
            dfs(node)
            if len(comp) > 0:
                comps.append(comp)
                comps_len.append(len(comp))

        print('largest connected component contains %d nodes' % max(comps_len))
```

```
largest connected component contains 40 nodes
```

In [9]:
```python
        ##################################################
        # 6                                              #
        ##################################################

        def cut_cost(s1, s2):
            cut = 0
            for edge in edges:
                x, y = edge
                if x in s1 and y in s2:
                    cut += 1

            cost = 0.5 * (cut/len(s1) + cut/len(s2))
            return cost

        comp = comps[comps_len.index(max(comps_len))]
        comp = list(comp)
        comp = sorted(comp)
        s1, s2 = comp[:len(comp)//2], comp[len(comp)//2:]

        cost = cut_cost(s1, s2)
        print('normalized-cut cost of the 50/50 split: %f' % cost)
```

```
normalized-cut cost of the 50/50 split: 4.600000
```

In [10]:
```python
         ##################################################
         # 7                                              #
         ##################################################
```

```
    def greedy(s1, s2, compute_cost):
        node_cost = []

        for node in s1:
            t1 = set(s1)
            t2 = set(s2)
            t1.remove(node)
            t2.add(node)
            node_cost.append((node, compute_cost(t1, t2)))

        for node in s2:
            t1 = set(s1)
            t2 = set(s2)
            t1.add(node)
            t2.remove(node)
            node_cost.append((node, compute_cost(t1, t2)))

        return node_cost
```

```
In [11]: comp = comps[comps_len.index(max(comps_len))]
         comp = list(comp)
         comp = sorted(comp)
         s1, s2 = set(comp[:len(comp)//2]), set(comp[len(comp)//2:])

         cur_cost = cut_cost(s1, s2)

         while(True):
             node_cost = greedy(s1, s2, cut_cost)
             node_cost = sorted(node_cost, key=lambda x:(x[1], x[0]))
             if node_cost[0][1] >= cur_cost: break

             cur_cost = node_cost[0][1]
             node = node_cost[0][0]
             if node in s1:
                 s1.remove(node)
                 s2.add(node)
             else:
                 s1.add(node)
                 s2.remove(node)

         print('Elements in split 1:\n' + str(s1))
         print('Elements in split 2:\n' + str(s2))
         print('normalized-cut cost: %f' % cur_cost)
```

```
Elements in split 1:
{769, 772, 774, 798, 800, 803, 805, 810, 811, 819, 823, 697, 828, 830, 703, 708, 840, 713, 719,
Elements in split 2:
```

```
{864, 804, 876, 893, 878, 882, 884, 888, 886, 729, 825, 889, 861, 863}
normalized-cut cost: 0.879121
```

```
In [12]: ##################################################
         # 8                                              #
         ##################################################

         def modularity(s):
             e = 0
             a = 0
             for edge in edges:
                 x, y = edge
                 if x in s and y in s:
                     e += 1
                 if x in s or y in s:
                     a += 1
             e /= len(edges)
             a /= len(edges)

             mod = e - a**2
             return mod

         def mod_cost(s1, s2):
             t = set(nodes) - s1 - s2
             return modularity(s1) + modularity(s2) + modularity(t)
```

```
In [13]: comp = comps[comps_len.index(max(comps_len))]
         comp = list(comp)
         comp = sorted(comp)
         s1, s2 = set(comp[:len(comp)//2]), set(comp[len(comp)//2:])

         cur_cost = mod_cost(s1, s2)

         while(True):
             node_cost = greedy(s1, s2, mod_cost)
             node_cost = sorted(node_cost, key=lambda x:(x[1], x[0]))
             if node_cost[0][1] >= cur_cost: break

             cur_cost = node_cost[0][1]
             node = node_cost[0][0]
             if node in s1:
                 s1.remove(node)
                 s2.add(node)
             else:
                 s1.add(node)
                 s2.remove(node)
```

```python
    print('Elements in split 1:\n' + str(s1))
    print('Elements in split 2:\n' + str(s2))
    print('modularity: %f' % cur_cost)
```

```
Elements in split 1:
{769, 772, 774, 804, 810, 811, 825, 697, 830, 703, 708, 840, 729, 863, 745, 876, 878, 880, 753,
Elements in split 2:
{798, 800, 803, 805, 819, 823, 828, 713, 719, 856, 861, 864, 869, 747, 882, 884, 886, 888, 889,
modularity: -0.416667
```

```
In [ ]:
```