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=0.717749
Test set accuracy=0.718229
```

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:12324        Negative:4342
TP:8997        TN:2973        FP:3327        :FN:1369
```

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

        # To assign more importance to FP, I increase the cost
        # (negative loglikelihood) by 10 when FP prediction occurs.

        def f2(theta, X, y, lam):
            loglikelihood = 0
            for i in range(len(X)):
                logit = inner(X[i], theta)
                t = log(1 + exp(-logit))
                if not y[i]:
                    t += logit

                # when FP occurs, increase loglikelihood by 10
                if y[i] == False and logit > 0:
                    t *= 10
                loglikelihood -= t

                for k in range(len(theta)):
                    loglikelihood -= lam * theta[k]*theta[k]

            return -loglikelihood

        def fprime2(theta, X, y, lam):
            dl = [0]*len(theta)
            for i in range(len(X)):
                logit = inner(X[i], theta)
                for k in range(len(theta)):
                    t = X[i][k] * (1 - sigmoid(logit))
                    if not y[i]:
                        t -= X[i][k]

                    # modify derivative accordingly
                    if y[i] == False and logit > 0:
                        t *= 10
                    dl[k] += t

            for k in range(len(theta)):
                dl[k] -= lam*2*theta[k]
            return numpy.array([-x for x in dl])

        def train2(lam, X, X_train, y_train):
            theta,_,_ = scipy.optimize.fmin_l_bfgs_b(f2, [0]*len(X[0]),
                                                     fprime2, pgtol = 10,
                                                     args = (X_train, y_train, lam))
            return theta
```

```
lam = 1.
theta = train2(lam, X, X_train, y_train)
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:0        Negative:16666
TP:0        TN:6300        FP:0        :FN:10366
```

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 = 0.010000
Train acc=0.720929        Validation acc=0.718409        Test acc=0.718829
```

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)
        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]: ##################################################
        # 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:]

        nodes = [n for n in nodes if n in comp]
        edges = [e for e in edges if e[0] in nodes or e[1] in nodes]
```

```
    nodes = set(nodes)
    edges = set(edges)

    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]:
```
##################################################
# 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):
    return modularity(s1) + modularity(s2)
```

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]), reverse=True)
```

```
            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('modularity: %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, 856, 825, 889, 861, 863}
modularity: 0.233368
```

In [ ]: