Machine Learning for Computer Vision

Exercise 3

Kodai Matsuoka, Yuyan Li

May 12, 2017

1 Linear Program

1.1 Implementation

Our program is shown in listing 1.
For the Linear Program we have to use the vector μ as introduced in the lecture. The function *givemu(x0,x1,x2)* produces such a vector for given (x0,x1,x2). For this system μ is a 18-component vector of the form:

$$\mu = \begin{pmatrix} \mu_0(0) & \mu_0(1) & \mu_1(0) & \mu_1(1) & \mu_2(0) & \mu_2(1) & \mu_{01}(0,0) & \mu_{01}(0,1) & \mu_{01}(1,0) \\ \mu_{01}(1,1) & \mu_{12}(0,0) & \mu_{12}(0,1) & \mu_{12}(1,0) & \mu_{12}(1,1) & \mu_{20}(0,0) & \mu_{20}(0,1) & \mu_{20}(1,0) & \mu_{20}(1,1) \end{pmatrix}$$

The function *coeff(p0,p1,p2,p01,p02,p12)* takes the potentials ψ_i and ψ_{ij} and puts them into a coefficient vector. It is also a 18-component vector of the form:

$$c = (\ \psi_0(0) \quad \psi_0(1) \quad \psi_1(0) \quad \psi_1(1) \quad \psi_2(0) \quad \psi_2(1) \quad \psi_{01}(0,0) \quad \psi_{01}(0,1) \quad \psi_{01}(1,0) \quad \psi_{01}(1,1)$$

$$\psi_{12}(0,0) \quad \psi_{12}(0,1) \quad \psi_{12}(1,0) \quad \psi_{12}(1,1) \quad \psi_{20}(0,0) \quad \psi_{20}(0,1) \quad \psi_{20}(1,0) \quad \psi_{20}(1,1) \ )$$

The constraint matrix A is chosen as:

$$A = \begin{pmatrix}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1
\end{pmatrix} \tag{1}$$

with the constraint (15-component) vector:

$$b = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

The constraint is thus:

$$A\mu = b$$

1.2 Attractive and Repulsive Potentials Part1

We use the given potentials to calculate μ for $\beta \in \{-1, 1\}$.
The output for $beta$ = +1.0

```
Optimization terminated successfully.
      Current function value: 1.100000
      Iterations: 13
beta= 1.0
solution vector mu=
[ 1.  0.  1.  0.  1.  0.  1.  0.  0.  0.  1.  0.  0.  0.  1.  0.  0.  0.]
result: [ 0.  0.  0.]
```

The output for $beta$ = -1.0

```
Optimization terminated successfully.
         Current function value: -1.900000
         Iterations: 16
beta= -1.0
solution vector mu=
[ 0.5  0.5  0.5  0.5  0.5  0.5  0.   0.5  0.5  0.   0.   0.5  0.5  0.   0.
0.5  0.5  0. ]
result: [ 0.  0.  0.]
```

Comment
1. We see that the program gives a non-integer result for $\beta = -1$. We need to use ILP to get the right solution.
2. For $\beta = 1$, the exact solutions (calculated by hand) are (x0,x1,x2)=(0,0,0) and (1,1,1). However it gave only one of these solutions which is (0,0,0). It also goes for $\beta = -1$.

Listing 1: exercise3.py

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import linprog

"""
I set solution vector "mu" as
mu=
[mu0(0),mu0(1),mu1(0),mu1(1),mu2(0),mu2(1)
,mu01(0,0),mu01(0,1),mu01(1,0),mu01(1,1)
,mu12(0,0),mu12(0,1),mu12(1,0),mu12(1,1)
,mu20(0,0),mu20(0,1),mu20(1,0),mu20(1,1)]
mui(k) = 1 (xi=k), 0 (otherwise)
muij(k,l) = 1 ((xi,xj)=(k,l)), 0 (otherwise)
"""
#######
# Implementation of Linear Program
#######

# give vector mu for given x0,x1,x2
def givemu(x0, x1, x2):
    mu = np.zeros(18)
    mu[x0] = 1
    mu[x1 + 2] = 1
    mu[x2 + 4] = 1

    mu[6] = mu[0]*mu[2]
    mu[7] = mu[0]*mu[3]
    mu[8] = mu[1]*mu[2]
    mu[9] = mu[1]*mu[3]

    mu[10] = mu[0]*mu[2]
    mu[11] = mu[0]*mu[3]
    mu[12] = mu[1]*mu[2]
    mu[13] = mu[1]*mu[3]

    mu[14] = mu[2]*mu[4]
    mu[15] = mu[2]*mu[5]
    mu[16] = mu[3]*mu[4]
    mu[17] = mu[3]*mu[5]
    return mu

# give vector x for given mu
```

```python
def givex(mu):
    x = np.zeros(3)
    x[0] = int(mu[1] == 1)
    x[1] = int(mu[3] == 1)
    x[2] = int(mu[5] == 1)
    return x

# for checking
def energy(cost, state):
    return sum(cost * state)


# coefficient vector
# pi are the unaries and pij are the pairwise factors
def coeff(p0, p1, p2, p01, p02, p12):
    c = np.zeros(18)
    c[0:2] = p0
    c[2:4] = p1
    c[4:6] = p2
    c[6:10] = p01.flatten()
    c[10:14] = p02.flatten()
    c[14:18] = p12.flatten()
    return c

# constraint matrix
A = np. zeros((15,18))
for i in range(3):
    A[i,2*i] = 1
    A[i,2*i+1] = 1

for i in range(3,15):
    j = i - 3
    k = j - 2*(j//4)
    A[i,k-6*(k//6)] = -1

a = np.array([[1,1,0,0],[0,0,1,1],[1,0,1,0],[0,1,0,1]])
A[3:7,6:10] = a
A[7:11,10:14] = a
A[11:15,14:18] = a

# constraint vector
b = np.zeros(15)
for i in range(3):
    b[i] = 1
```

```
bounds = (0, None)

######
# Attractive and Repulsive Potentials
######

# coeff vector of given system
# choose beta
beta = +1.0
p0 = [.1, .1]
p1 = [.1, .9]
p2 = [.9, .1]
pp = np.array([[0., beta],[beta,0.]])
c = coeff(p0,p1,p2,pp,pp,pp)

res = linprog(c, A_eq=A, b_eq=b, bounds=(bounds), options={"
    disp": True})
x_res = givex(res.x)

print("beta=",beta)
# print("coefficients vector c=\n",c)
# print("constraint matrix A=\n",A)
# print("constraint vector b=\n",b)
print("solution vector mu=\n",res.x)
print(f"result: {x_res}")

# some checks
# print(energy(c, res.x))
# print(energy(c, givemu(0,0,0)))
# print(givemu(1,1,1))
# print(energy(c, givemu(1,1,1)))
```