ואדים ליטבינוב

314552365

תרגיל 4 – למידת מכונה סטטיסטית

דגימות ה-x שנוצרו:

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

דגימות ה-y שנוצרו:

```
[[ 3.03365055, -1.62012621, -0.29964224,  3.83618757,  0.66738074]
 [ 2.20489206,  0.79584422,  2.76882266,  2.2065719,   1.86206922]
 [ 0.18228087,  1.23864598,  2.41689575,  0.06448298,  0.87737369]
 [ 1.00968418, -0.02321985,  1.61560156,  0.6711655,  -0.83726598]
 [ 2.0800301,  -0.95275016,  0.24920473,  0.56620424,  2.13844412]]
```

הסתברויות מדויקות:

```
[[0.99633942, 0.97143201, 0.98794063, 0.99972909, 0.98402649]
 [0.99927409, 0.99927676, 0.99992839, 0.99992675, 0.99865348]
 [0.99594175, 0.99972107, 0.99994069, 0.99929021, 0.99689055]
 [0.99701466, 0.99828765, 0.99970424, 0.99900649, 0.98686669]
 [0.9925306,  0.98364023, 0.9933976,  0.99564039, 0.9932133 ]]
```

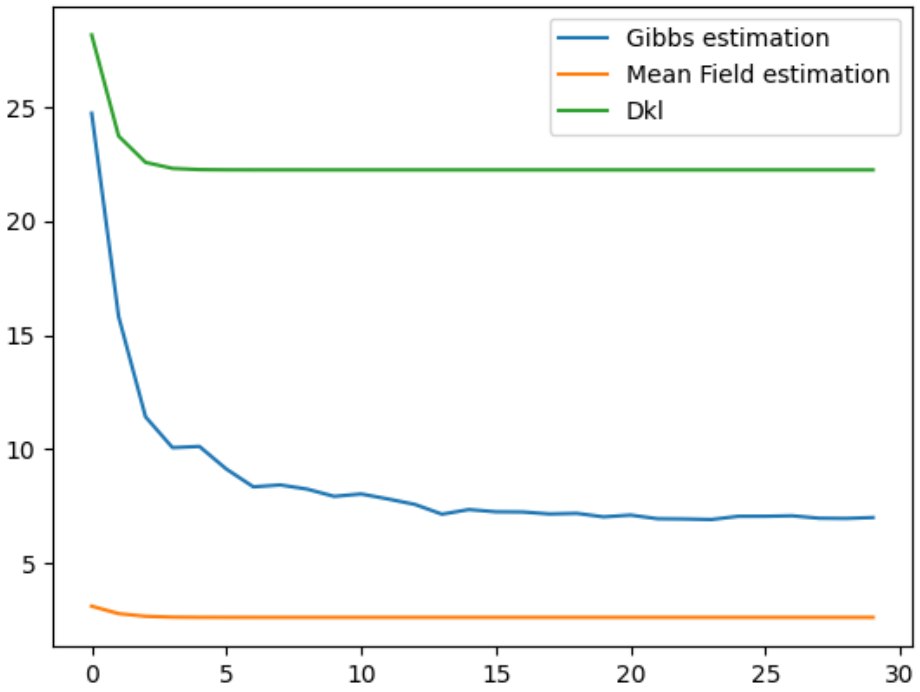
שערוכי ההסתברות של אלגוריתם גיבס:

[[0.53333333, 0.36666667, 0.46666667, 0.6, 0.46666667]
[0.43333333, 0.5, 0.36666667, 0.36666667, 0.43333333]
[0.46666667, 0.56666667, 0.53333333, 0.33333333, 0.6]
[0.56666667, 0.36666667, 0.56666667, 0.56666667, 0.46666667]
[0.56666667, 0.5, 0.43333333, 0.33333333, 0.4]]

שערוכי ההסתברויות של אלגוריתם Mean Field:

[[0.24779231, 0.1588464, 0.57688601, 0.13898384, 0.72750744]
[0.79859936, 0.94101579, 0.85670781, 0.89383092, 0.89834685]
[0.9146376, 0.97637628, 0.94084516, 0.96435306, 0.92923305]
[0.93097883, 0.95357423, 0.97297985, 0.9714811, 0.71978624]
[0.73365833, 0.66551715, 0.90736282, 0.92520159, 0.73046052]]

הגרף המבוקש:



הקוד:

```
# Vadim Litvinov
import numpy as np
from itertools import product
import matplotlib.pyplot as plt

NUM_ITER = 1000
N_ROWS = 5
N_COLUMNS = 5
T = 30


def generateData():
    x = np.random.randint(2, size=(N_ROWS, N_COLUMNS))
    for t in range(NUM_ITER):
        #print(x)
        sum_neighbor_mat = sumMarkovBlankets(x)
        #print(sum_neighbor_mat)
        for i in range(x.shape[0]):
            for j in range(x.shape[1]):
                sum_phi = [0, 0]
                for a in [0, 1]:
                    sum_phi[a] += verticalSum(i, j, x, a)
                #print(sum_phi)
                p_for_1 = np.exp(sum_phi[1]) / np.sum(np.exp(sum_phi))
                #print(np.exp(sum_phi))
                x[i][j] = np.random.binomial(1, p_for_1, 1)
                #x[i][j] = np.random.choice(2, 1, p=[1-p_for_1, p_for_1])
    y = np.random.normal(x, 1)
    print(x)
    print(y)
    return x, y
    # end of part A


def isInside(x, y, m):
    return 0 <= x < len(m[0]) and 0 <= y < len(m)


def verticalPSum(x, y, m, a):
    s = 0
    if isInside(x, y - 1, m):
        if a == 1:
            s += m[x][y-1]
        else:
            s += 1 - m[x][y-1]
    if isInside(x, y + 1, m):
        if a == 1:
            s += m[x][y + 1]
        else:
            s += 1 - m[x][y + 1]
    if isInside(x - 1, y, m):
        if a == 1:
            s += m[x - 1][y]
        else:
            s += 1 - m[x - 1][y]
    if isInside(x + 1, y, m):
        if a == 1:
            s += m[x + 1][y]
        else:
            s += 1 - m[x + 1][y]
    return s


def verticalSum(x, y, m, a):
    return sum([(isInside(x, y - 1, m) and m[x][y - 1] == a),
                (isInside(x, y + 1, m) and m[x][y + 1] == a),
                (isInside(x - 1, y, m) and m[x - 1][y] == a),
                (isInside(x + 1, y, m) and m[x + 1][y] == a)])
```

```
def sumMarkovBlankets(m):
    vertical = []
    for i in range(len(m)):
        vertical.append([])
        for j in range(len(m[0])):
            vertical[i].append(verticalSum(i, j, m, m[i, j]))
    return np.asarray(vertical)


def computeExactMarginals(y):
    #compute p(x=1|y)
    z_y = 0
    p_xy = np.zeros((N_ROWS, N_COLUMNS))
    #for i in range(2**(N_ROWS*N_COLUMNS)):
    for comb in product([0, 1], repeat=(N_ROWS*N_COLUMNS)):
        exp_sums, sum_edges, sum_nodes = 0, 0, 0
        comb = np.asarray(comb).reshape(N_ROWS, N_COLUMNS)
        #comb = np.array(comb).reshape(y.shape[0], y.shape[1])
        #print(comb)
        for i in range(N_ROWS):
            for j in range(N_COLUMNS):
                sum_edges += verticalSum(i, j, comb, comb[i][j])
                sum_nodes += -0.5*np.power(comb[i][j] - y[i][j], 2)
                #exp_sums = np.exp(sum_edges + sum_nodes)
        z_y += np.exp(sum_edges + sum_nodes)
        for i in range(N_ROWS):
            for j in range(N_COLUMNS):
                if comb[i][j] == 1:
                    #exp_sums = np.exp(sum_edges + sum_nodes)
                    p_xy[i][j] += np.exp(sum_edges + sum_nodes)
    p_xy = p_xy / z_y
    print(p_xy)
    return p_xy


def gibbs(y):
    x_sum = np.zeros(shape=(N_ROWS, N_COLUMNS))
    p_xy = np.zeros(shape=(T, N_COLUMNS, N_ROWS))
    x_estimat = np.random.randint(2, size=(N_ROWS, N_COLUMNS))
    #gibbs iterations
    for t in range(1, T):
        sum_edges, sum_nodes = 0, 0
        #loop over the grid
        for i in range(N_ROWS):
            for j in range(N_COLUMNS):
                ij_sum = [0, 0]
                for a in range(2):
                    sum_edges += verticalSum(i, j, x_estimat, x_estimat[i][j])
                    sum_nodes += -0.5 * np.power(x_estimat[i][j] - y[i][j], 2)
                    ij_sum += (sum_edges + sum_nodes)
                p = np.exp(ij_sum[1]) / (np.sum(np.exp(ij_sum)))
                x_estimat[i][j] = np.random.binomial(1, p, 1)
                if x_estimat[i][j] == 1:
                    x_sum[i][j] += x_estimat[i][j]
                #careful not to divide by 0
                p_xy[t][i][j] = x_sum[i][j] / (t+1)
    print(p_xy[T-1])
    return p_xy


def meanFieldApproximation(y):
    q = np.random.uniform(0, 1, size=(N_ROWS, N_COLUMNS))
    p_xy = np.zeros((T, N_ROWS, N_COLUMNS))
    #loop until convergence
    for t in range(0, T):
        sum_edges, sum_nodes = 0, 0
        for i in range(0, N_ROWS):
            for j in range(0, N_COLUMNS):
                ij_sum = [0, 0]
                for a in range(2):
```

```
                    sum_edges += verticalPSum(i, j, q, a)
                    # this time we need to use both a values
                    sum_nodes += -0.5 * np.power(a - y[i][j], 2)
                    ij_sum[a] += (sum_edges + sum_nodes)
                q[i][j] = np.exp(ij_sum[1]) / (np.sum(np.exp(ij_sum)))
                #print(q[i][j])
                p_xy[t][i][j] = q[i][j]
    print(p_xy[T-1])
    return p_xy


def plotEst(p_correct, p_gibbs , p_mf, kl):
    error_mf = np.power(np.add(p_mf, -p_correct), 2)
    error_gibbs = np.power(np.add(p_gibbs, -p_correct), 2)
    err_mf = np . sum(np.sum(error_mf, axis =2), axis=1)
    err_gibbs = np.sum(np.sum(error_gibbs , axis =2), axis=1)
    plt.plot(range(len(err_gibbs)), err_gibbs , label='Gibbs estimation ')
    plt. plot(range(len(err_mf)), err_mf, label='Mean Field estimation')
    plt.plot(range(len(kl)), kl, label='Dkl')
    plt.legend()
    plt.show()


def DKL(p, q):
    kl_div = np.zeros(shape=T)
    for t in range(T):
        #for a in [0, 1]:
        for i in range(N_ROWS):
            for j in range(N_COLUMNS):
                dkl_1 = q[t][i][j] * np.log(q[t][i][j] / p[i][j])
                dkl_0 = (1 - q[t][i][j]) * np.log((1 - q[t][i][j]) / (1 -
p[i][j]))
                kl_div[t] += dkl_1
                kl_div[t] += dkl_0
    return kl_div


if __name__ == '__main__':
    x, y = generateData()
    p_cor = computeExactMarginals(y)
    p_gibbs = gibbs(y)
    p_mean_field = meanFieldApproximation(y)
    kl = DKL(p_cor, p_mean_field)
    print(kl)
    plotEst(p_cor, p_gibbs, p_mean_field, kl)
```