CSE 291-D: Homework 4

Hao-en Sung [A53204772] (wrangle1005@gmail.com)
Department of Computer Science
University of California, San Diego
San Diego, CA 92092

Problem 1

(a)

According to Markov Random Field, the conditional probability of x_i given all the other variables can be simplified as follows.

$$\begin{aligned}
p(x_i|\boldsymbol{x} \setminus x_i) &= p(x_i|\text{nb}(x_i)) \\
&= \frac{p(x_i, \text{nb}(x_i))}{p(\text{nb}(x_i))} \\
&= \frac{\frac{1}{Z} e^{\sum_{j\in\text{nb}(x_i)} w_{i,j} x_i x_j + b x_i + \sum_{j\in\text{nb}(x_i)} b x_j}}{\frac{1}{Z} e^{\sum_{j\in\text{nb}(x_i)} b x_j}} \\
&= e^{\sum_{j\in\text{nb}(x_i)} w_{i,j} x_i x_j + b x_i}
\end{aligned}$$

(b)

It can be derived that

$$\begin{aligned}
q_i^+ &= \frac{e^{+1\cdot a_i}}{e^{+1\cdot a_i} + e^{-1\cdot a_i}} \\
&= \frac{1}{1 + e^{-2\cdot a_i}} \\
&= \text{sigm}(2a_i).
\end{aligned}$$

(c)

Based on the definition of μ, I have

$$\begin{aligned}
\mu_i = \mathbb{E}_{q_i}[x_i] = \sum_{q_i} q_i x_i \\
&= +1 \cdot \text{sigm}(2a_i) + (-1) \cdot \text{sigm}(-2a_i) \\
&= \frac{1}{1 + e^{-2a_i}} - \frac{1}{1 + e^{2a_i}} \\
&= \frac{e^{a_i} - e^{-a_i}}{e^{a_i} + e^{-a_i}} \\
&= \tanh(a_i)
\end{aligned}$$

(d)

If I focus ELBO boundary on only factor i, I have

$$\begin{aligned}
\text{ELBO} &= \mathcal{L}(\boldsymbol{q}) = \mathbb{E}_{q_i}\left[\log p(\boldsymbol{x},\boldsymbol{y}) - \log q(\boldsymbol{x})\right] \\
&= \mathbb{E}_{q_i}\left[\log p(\boldsymbol{x}|\boldsymbol{y}) - \log q(\boldsymbol{x})\right] + \log p(\boldsymbol{y}) \\
&= \sum_{x_i} \left\{ q_i(x_i)\cdot \mathbb{E}_{\boldsymbol{q}\setminus q_i}\left[\log p(\boldsymbol{z},\boldsymbol{x})\right]\right\} - \mathbb{E}_{q_i}\left[\log q(\boldsymbol{x})\right] + \text{const} \\
&= \mathbb{E}_{q_i}\left[\log p(\boldsymbol{x}|\boldsymbol{y}) - \log q(\boldsymbol{x})\right] + \log p(\boldsymbol{y}) \\
&= \sum_{x_i} \left\{ q_i(x_i)\cdot \mathbb{E}_{\boldsymbol{q}\setminus q_i}\left[\frac{1}{2}\sum_{i,j} w_{i,j}x_i x_j + \sum_i b_i x_i\right]\right\} - \mathbb{E}_{q_i}\left[\log q(\boldsymbol{x})\right] + \text{const} \\
&= \sum_{x_i} \left\{ q_i(x_i)\cdot x_i \left[\sum_{j\in \text{nb}(x_i)} w_{i,j}\mu_j + b_i\right]\right\} - \mathbb{E}_{q_i}\left[\log q(\boldsymbol{x})\right] + \text{const} \\
&= q_i^+(x_i)\left[\sum_{j\in \text{nb}(x_i)} w_{i,j}\mu_j + b_i\right] - \left(1 - q_i^+(x_i)\right)\cdot x_i \left[\sum_{j\in \text{nb}(x_i)} w_{i,j}\mu_j + b_i\right] \\
&\qquad - q(\boldsymbol{x})\log q(\boldsymbol{x}) - (1-q(\boldsymbol{x}))\log(1-q(\boldsymbol{x})) + \text{const} \\
&= (2q_i^+(x_i) - 1)\left[\sum_{j\in \text{nb}(x_i)} w_{i,j}\mu_j + b_i\right] - q(\boldsymbol{x})\log q(\boldsymbol{x}) - (1-q(\boldsymbol{x}))\log(1-q(\boldsymbol{x})) + \text{const}
\end{aligned}$$

(e)

According to the derivation on textbook, I know that

$$\begin{aligned}
\mathcal{L}(\boldsymbol{q}) &= \sum_{x_i} q_i(x_i)\log f_i(x_i) + H(q_i) + \text{const} \\
&= -D_{KL}(q_i(x_i)||f_i(x_i)) + \text{const.}
\end{aligned}$$

If jointly considering the conclusion from (d) and textbook formula, $\mathcal{L}(\boldsymbol{q})$ can be maximized when

$$\begin{aligned}
q_i(x_i) &= f_i(x_i) \\
e^{a_i x_i} &= e^{x_i\left[\sum_{j\in \text{nb}(x_i)} w_{i,j}\mu_j + b_i\right]} \\
a_i &= \sum_{j\in \text{nb}(x_i)} w_{i,j}\mu_j + b_i.
\end{aligned}$$

Thus, I derive the update rule for a_i.

Problem 2

(a)

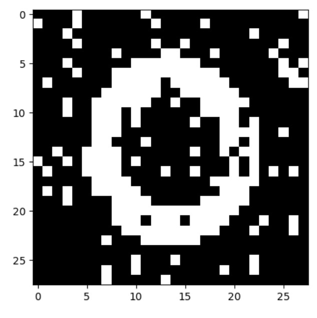
As what I derived in previous problem, I directly calculate the probability of $p(x_i = +1|\text{nb}(x_i))$ and $p(x_i = -1|\text{nb}(x_i))$ separately and then normalize them using log-sum-exp trick. I run gibbs sampling for 10 times and then calculate the average result.

The original noisy picture and denoised picture are shown as Fig. 1.

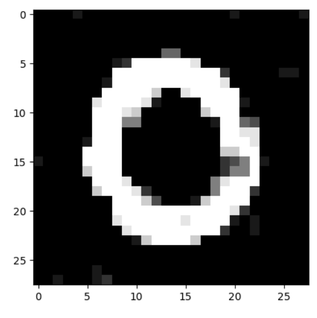
The average accuracy after 10 rounds Gibbs sampling are listed as follows.

Perturbed Pixels:	0.820513
Unperturbed Pixels:	0.991501
Overall:	0.974490

My implementation is written in Python, shown as Code 1 in Appendix.



(a) Noisy Picture



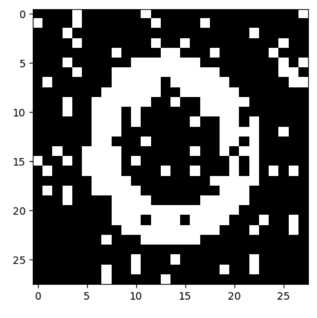
(b) Denoised Picture

Figure 1: Gibbs Sampling

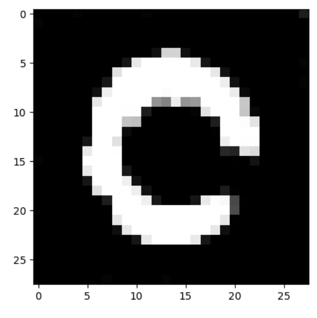
(b)

Similar to (a), I directly implement the conclusion from previous problems. First, I uniformly generate $\mu_i \in [-1, 1]$. Later, I iteratively update each a_i and μ_i for 10 rounds.

The original noisy picture and denoised picture are shown as Fig. 2.



(a) Noisy Picture



(b) Denoised Picture

Figure 2: Mean Field Algorithm

The average accuracy after 10 rounds Gibbs sampling are listed as follows.

$$\begin{aligned} \text{Perturbed Pixels: } & 0.858974 \\ \text{Unperturbed Pixels: } & 0.974504 \\ \text{Overall: } & 0.963010 \end{aligned}$$

My implementation is written in Python, shown as Code 2 in Appendix.

(c)

Due to the time factor, unfortunately, I decide to skip this problem in this homework.

Problem 3

(a)

In my model, I use the following as observed predicates

- Friendship [Friends(X, Y)]

- Joining Democratic events [DNC-E(X)]
- Joining Republican events [GOP-E(X)]
- Watching Democratic news [DNC-N(X)]
- Watching Republican news [GOP-N(X)],

while regarding

- Democratic Preference [DNC-P(X)]
- Republican Preference [GOP-P(X)]

as unobserved predicates.

There is no latent variable in my model design.

(b), (c)

The Markov logic rules for the above-mentioned predicates are listed below.

$$\begin{aligned}
5.0 &: \quad \forall X \forall Y \forall Z \text{ Friends}(X,Y) \wedge \text{Friends}(Y,Z) \rightarrow \text{Friends}(Y,Z) \\
3.0 &: \quad \forall X \forall Y \text{ Friends}(X,Y) \wedge \text{DNC-E}(X) \rightarrow \text{DNC-E}(Y) \\
3.0 &: \quad \forall X \forall Y \text{ Friends}(X,Y) \wedge \text{GOP-E}(X) \rightarrow \text{GOP-E}(Y) \\
1.0 &: \quad \forall X \forall Y \text{ Friends}(X,Y) \wedge \text{DNC-N}(X) \rightarrow \text{DNC-N}(Y) \\
1.0 &: \quad \forall X \forall Y \text{ Friends}(X,Y) \wedge \text{GOP-N}(X) \rightarrow \text{GOP-N}(Y) \\
10.0 &: \quad \forall X \text{ DNC-E}(X) \rightarrow \text{DNC-P}(X) \\
10.0 &: \quad \forall X \text{ GOP-E}(X) \rightarrow \text{GOP-P}(X) \\
2.0 &: \quad \forall X \text{ DNC-N}(X) \rightarrow \text{DNC-P}(X) \\
2.0 &: \quad \forall X \text{ GOP-N}(X) \rightarrow \text{GOP-P}(X)
\end{aligned}$$

(d)

The data collection is straightforward. Features, including friendship, events attendance, and news selection, can be collected through BookFace. In order to update the weighting of our model, we also need to have some ground truths, i.e. a group of people with clear political preference. After training process, we can run on a validation set to measure our model performance. At the end, the prediction on rest of the data can be done.

In terms of the learning procedure, I can first write down the joint probability of my model as

$$P(\boldsymbol{x}) = \frac{1}{Z} e^{\sum_i w_i n_i(\boldsymbol{x})},$$

where $n_i(\boldsymbol{x})$ indicates the number of true groundings of one of my rules listed above.

With this joint probability function, I can easily update the weight of each formula by calculating the derivative of the log-likelihood, such as

$$\frac{\partial}{\partial w_i} \log P_w(\boldsymbol{X} = \boldsymbol{x}) = n_i(\boldsymbol{x}) - \sum_{\boldsymbol{x}'} P_w(\boldsymbol{X} = \boldsymbol{x}') n_i(\boldsymbol{x}'),$$

then apply stochastic gradient descent to iteratively update the model until convergence.

Appendix

Code Listing 1: Gibbs Sampling for Ising Model

```
import matplotlib.pyplot as plt
from matplotlib import cm
import pandas as pd
import numpy as np
import math


# fix numpy seed
np.random.seed(514)
TARGET = 1
THRESHOLD = 128


# read data
df = pd.read_csv('../dat/train.csv', sep=',')

# pick up one image
lab = df.iloc[TARGET,0]
img = df.iloc[TARGET,1:].as_matrix()
img = img.astype(np.uint8)

# set threshold for images
img[img < THRESHOLD] = 0
img[img >= THRESHOLD] = 255

# build ground truth matrix into {-1, +1}
gmat = img.astype(np.int64)
gmat = gmat / 255 * 2 - 1

# flip 10% at random
idx = np.random.choice(range(784), 78, replace=False)
rev_idx = list(np.setdiff1d(np.array(range(784)), idx))
img[idx] = 255-img[idx]

# reshape array into image
img = img.reshape((28,28))
plt.imshow(img,cmap=cm.gray)
plt.savefig('../res/origin.png')

# map matrix into {-1, +1}
mat = img.astype(np.int64)
mat = mat / 255 * 2 - 1


# accuracy calculation
def calACC(pmat, gmat, idx):
    ret = 0.0
    pmax = pmat.flatten() >= 0
    gmat = gmat >= 0

    for i in idx:
        if pmax[i] == gmat[i]:
            ret += 1.0
    return ret / len(idx)


# Gibbs Sampler Implementation
NUM_ITER = 10
CONST = 1

bmat = mat * CONST
```

```
tmat = np.zeros((28,28))

for t in range(NUM_ITER):
    for i in range(28):
        for j in range(28):
            p0 = -bmat[i][j]
            p1 = +bmat[i][j]
            if i > 0:
                p0 += -1 * mat[i-1][j]
                p1 += +1 * mat[i-1][j]
            if i < 27:
                p0 += -1 * mat[i+1][j]
                p1 += +1 * mat[i+1][j]
            if j > 0:
                p0 += -1 * mat[i][j-1]
                p1 += +1 * mat[i][j-1]
            if j < 27:
                p0 += -1 * mat[i][j+1]
                p1 += +1 * mat[i][j+1]
            p0 = p0
            p1 = p1

            if p0 < p1:
                ep1 = math.exp(p1-p0) / (1 + math.exp(p1-p0))
            else:
                ep1 = 1.0 / (1 + math.exp(p0-p1))

            nv = np.random.choice([-1, +1], 1, p=[1-ep1, ep1])
            mat[i][j] = nv

    tmat += mat

# average the result
tmat /= 10

# calculate accuracy
print 'Perturbed Pixels: %f' % calACC(tmat, gmat, idx)
print 'Unperturbed Pixels: %f' % calACC(tmat, gmat, rev_idx)
print 'Overall: %f' % calACC(tmat, gmat, range(784))

# map matrix into {0, 255}
tmat = (tmat + 1) / 2 * 255
img = tmat.astype(np.uint8)
plt.imshow(img,cmap=cm.gray)
plt.savefig('../res/gibbs_sampling.png')
```

Code Listing 2: Mean Field Algorithm for Ising Model

```
import matplotlib.pyplot as plt
from matplotlib import cm
import pandas as pd
import numpy as np
import math


# fix numpy seed
np.random.seed(514)
TARGET = 1
THRESHOLD = 128


# read data
df = pd.read_csv('../dat/train.csv', sep=',')

# pick up one image
```

```
lab = df.iloc[TARGET,0]
img = df.iloc[TARGET,1:].as_matrix()
img = img.astype(np.uint8)

# set threshold for images
img[img < THRESHOLD] = 0
img[img >= THRESHOLD] = 255

# flip 10% at random
idx = np.random.choice(range(784), 78, replace=False)
img[idx] = 255-img[idx]

# reshape array into image
img = img.reshape((28,28))
plt.imshow(img,cmap=cm.gray)

# map matrix into {-1, +1}
mat = img.astype(np.int64)
mat = mat / 255 * 2 - 1


# Mean Field Algorithm Implementation
NUM_ITER = 10
CONST = 1

bmat = mat * CONST
mmat = np.random.uniform(-1.0, +1.0, (28,28))

for t in range(NUM_ITER):
    for i in range(28):
        for j in range(28):
            a = bmat[i,j]
            if i > 0:
                a += mmat[i-1,j]
            if i < 27:
                a += mmat[i+1,j]
            if j > 0:
                a += mmat[i,j-1]
            if j < 27:
                a += mmat[i,j+1]
            mmat[i,j] = np.tanh(a)
mmat = (mmat + 1) / 2 * 255
img = mmat.astype(np.uint8)
plt.imshow(img,cmap=cm.gray)
```