CS561 HW07

October 23, 2023

1 Problem 1

Let $\mathbf{x} \in \mathbb{R}^n \sim f(\mathbf{x})$ be a vector of continuous random variables, and let y be a discrete random variable. By Bayes' rule,

$$\mathbb{P}(y = y_0 | \mathbf{x} \in A) = \frac{\mathbb{P}(\mathbf{x} \in A | y = y_0)\mathbb{P}(y = y_0)}{\mathbb{P}(\mathbf{x} \in A)}$$

for any $A \subseteq \mathbb{R}^n$ and value y_0 that y takes. Denote by $\mathbf{x}_0 = (x_1, x_2, \dots, x_n)$. Suppose $A_\delta = [x_1, x_1 + \delta] \times [x_2, x_2 + \delta] \times \cdots \times [x_n, x_n + \delta]$ is a region of $\mathbb{R}^n$. Consider the limit

$$\begin{aligned}
\lim_{\delta \to 0} \mathbb{P}(y = y_0 | \mathbf{x} \in A_\delta) &= \lim_{\delta \to 0} \frac{\mathbb{P}(\mathbf{x} \in A_\delta | y = y_0)\mathbb{P}(y = y_0)}{\mathbb{P}(\mathbf{x} \in A_\delta)} \\
p(y_0 | \mathbf{x}_0) &= \mathbb{P}(y = y_0) \lim_{\delta \to 0} \frac{\mathbb{P}(\mathbf{x} \in A_\delta | y = y_0)}{\mathbb{P}(\mathbf{x} \in A_\delta)} \\
&= \mathbb{P}(y = y_0) \frac{\lim_{\delta \to 0} \frac{\mathbb{P}(\mathbf{x} \in A_\delta | y = y_0)}{\delta^n}}{\lim_{\delta \to 0} \frac{\mathbb{P}(\mathbf{x} \in A_\delta)}{\delta^n}}
\end{aligned}$$

By the continuity of f, we can argue that

$$f(\mathbf{x_0}) = \frac{\partial^n}{\partial x_1 \dots \partial x_n} \mathbb{P}(\mathbf{x} \in A) = \left(\lim_{\delta \to 0} \frac{\mathbb{P}(\mathbf{x} \in A_\delta)}{\delta^n} \right)_{(\mathbf{x} = \mathbf{x_0})}$$

therefore,

$$\begin{aligned}
p(y_0 | \mathbf{x}_0) &= \mathbb{P}(y = y_0) \frac{\lim_{\delta \to 0} \frac{\mathbb{P}(\mathbf{x} \in A_\delta | y = y_0)}{\delta^n}}{\lim_{\delta \to 0} \frac{\mathbb{P}(\mathbf{x} \in A_\delta)}{\delta^n}} \\
&= p(y_0) \frac{f(\mathbf{x_0} | y_0)}{f(\mathbf{x_0})}
\end{aligned}$$

for any $x_0 \in \mathbb{R}^n$ and for any y_0 that y can take, as desired.

2 Problem 2

Consider a feature vector $\mathbf{x} \in \mathbb{R}^2$ and $y \in \mathbb{R}$ with joint pdf

$$p(\mathbf{x}, y) = \begin{cases} 6 & x_1, x_2, y \geq 0 \text{ and } x_1 + x_2 + y \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

assuming $\mathbf{x} = (x_1, x_2)$. Suppose $f : \mathbb{R}^2 \to \mathbb{R}$

(a) Let f minimize $\mathbb{E}[\ell(f(\mathbf{x}), y)]$. We can derive

$$\mathbb{E}[\ell(f(\mathbf{x}), y)] = \int_0^1 \int_0^{x_1} \int_0^{1-x_1-x_2} (f(x_1, x_2) - y)^2 p(x_1, x_2, y) dy dx_2 dx_1$$

which means

$$\int_0^1 \int_0^{x_1} \int_0^{1-x_1-x_2} (f(x_1, x_2) - y)^2 dy dx_2 dx_1$$

attains a minimum. If, for each x_1, x_2, the value (per x_1, x_2)

$$\int_0^{1-x_1-x_2} (f(x_1, x_2) - y)^2 dy$$

always attains a minimum, then so does the entire integral. Since x_1, x_2 are both fixed, we can treat $f(x_1, x_2)$ as a constant c (similarly, denote by $a = 1 - x_1 - x_2$) in which we can compute

$$\begin{aligned}
\int_0^{1-x_1-x_2} (f(x_1, x_2) - y)^2 dy &= \int_0^a (c - y)^2 dy \\
&= \int_0^a (c^2 + y^2 - 2cy) dy \\
&= \frac{a \cdot (3c^2 - 3ac + a^2)}{3}
\end{aligned}$$

the value is minimal when its derivate w.r.t. c is zero, which is attained when $6c - 3a = 0$. Therefore, $c = \frac{a}{2} = \frac{1-x_1-x_2}{2}$. The function that attains this minima is

$$f(x_1, x_2) = \frac{1 - x_1 - x_2}{2}$$

(b) The minimum risk can be evaluated from the integral

$$\int_0^1 \int_0^{x_1} \int_0^{1-x_1-x_2} 6 \left(\frac{1 - x_1 - x_2}{2} - y \right)^2 dy dx_2 dx_1$$

3 Problem 3

(a) Imagine the pixels are enumerated so that the x_1 is the pixel in the upper left corner of the image, x_2 is the pixel below x_1 (in the first column and second row) and so on. Pixel 34 would have two parents in the DAG: pixel 33 (top) and pixel 6 (left). Therefore,

$$p(x_{34} | x_1, x_2, \dots, x_{33}) = p(x_{34} | x_6, x_{33})$$

(b) Assuming the same configuration as above, one can roughly estimate (ignoring edge cases)

$$p(x_n | x_1, x_2, \dots, x_{n-1}) = p(x_n | x_{n-28}, x_{n-1})$$

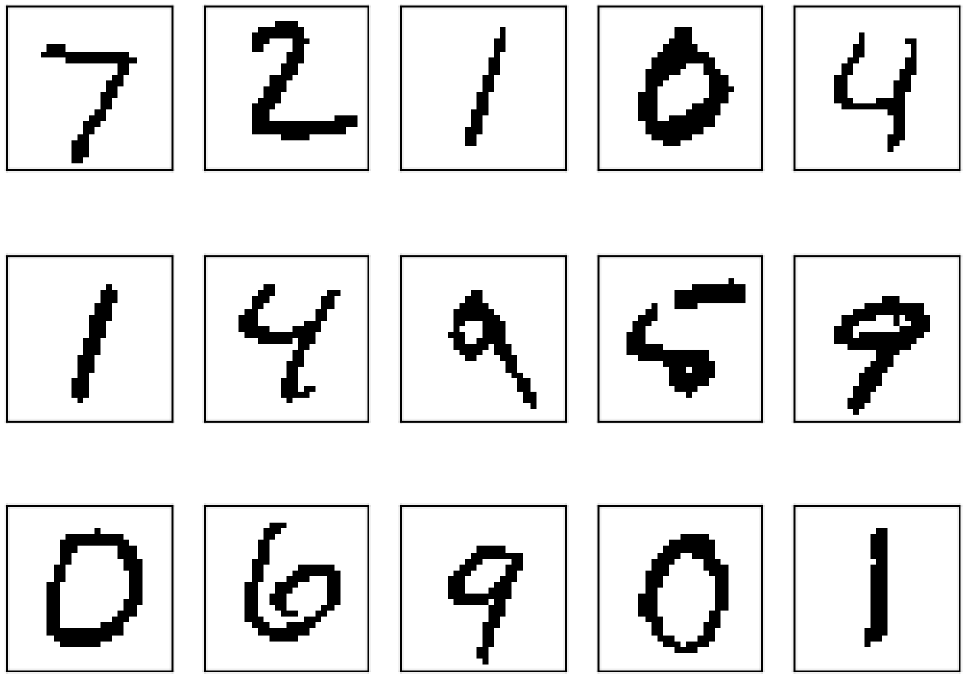
when $n - 1$ is the pixel to the top, and $n - 28$ is the pixel to the right. Therefore,

$$p(\mathbf{x} | y) = \prod_{n=1}^{784} p(x_n | x_{n-28}, x_{n-1}, y)$$

(c) The product suggests that, for each class y, each pixel has 4 values to model (its two parent pixels, each taking 2 values). The total number of parameter would be $y(4n) = 31360$. Precise calculation (with consideration of edge cases) would end up with 30250 parameters in total. Both numbers are greater than naive independence assumption ($yn = 7680$ parameters) but less than taking account of every possible configuration ($2^n = 2^{784}$).

[7]:
```
import numpy as np
import warnings
warnings.filterwarnings("ignore")

import tensorflow as tf
import matplotlib.pyplot as plt
from matplotlib import pyplot as plt

(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

## function to plot images in grid
def show_images(images, rows, cols):
    for i in range(rows * cols):
        plt.subplot(rows, cols, i + 1)
        plt.imshow(images[i], cmap=plt.cm.gray_r)
        plt.xticks(())
        plt.yticks(())
    plt.show()

# convert to 0/1 (instead of 0-255)
x_train_int = np.array([np.round(1.0*i/256) for i in x_train])
x_test_int = np.array([np.round(1.0*i/256) for i in x_test])

## Uncomment below to see a few images
print('A few example images:')
show_images(x_test_int, 3, 5)
```

```
A few example images:
```



[8]:

```python
#3d, assuming edge cases is surrouned by 0,0
import itertools
count = np.zeros([2,10,4,28,28])
# 2 values, 0 vs 1
# 10 classes
# 4 prior values, (0,0) (0,1) (1,0) (1,1)
# 28x28 pixels

for y in range(10):
    x_conditioned = np.zeros([sum(y_train==y),29,29])
    x_conditioned[:,1:,1:] = x_train_int[y_train==y]
    for k in range(sum(y_train==y)):
        for i,j in itertools.product(list(range(28)), list(range(28))):
            value = int(x_conditioned[k,i+1,j+1])
            parents_index = int(x_conditioned[k,i,j+1]*2+x_conditioned[k,i+1,j])
            count[value,y,parents_index,i,j]+=1

total = np.expand_dims(count.sum(axis=0), 0)
probabilities = (count+1)/(total+2)
```

[3]:

```python
#3e, assuming p(y) is uniform, we maximize p(x|y)
def log_likelihood(image, y):
    log_prob = 0
    image_padded = np.zeros([29,29])
```

```
    image_padded[1:,1:] = image
    for i,j in itertools.product(list(range(28)), list(range(28))):
        value = int(image_padded[i+1,j+1])
        parents_index = int(image_padded[i,j+1]*2+image_padded[i+1,j])
        log_prob += np.log(probabilities[value,y,parents_index,i,j])
    return log_prob
```

[4]:
```
#3f
from tqdm import tqdm
def ml_predict(image):
    return np.argmax([log_likelihood(image, y) for y in range(10)])
y_test_pred = [ml_predict(x) for x in tqdm(x_test_int)]
```

```
100%|                                      |
10000/10000 [02:23<00:00, 69.59it/s]
```

[5]:
```
#3f (cont)
print(f"Error rate = {np.sum(y_test_pred!=y_test)/len(y_test)}")
print(f"Accuracy = {np.sum(y_test_pred==y_test)/len(y_test)}")
```

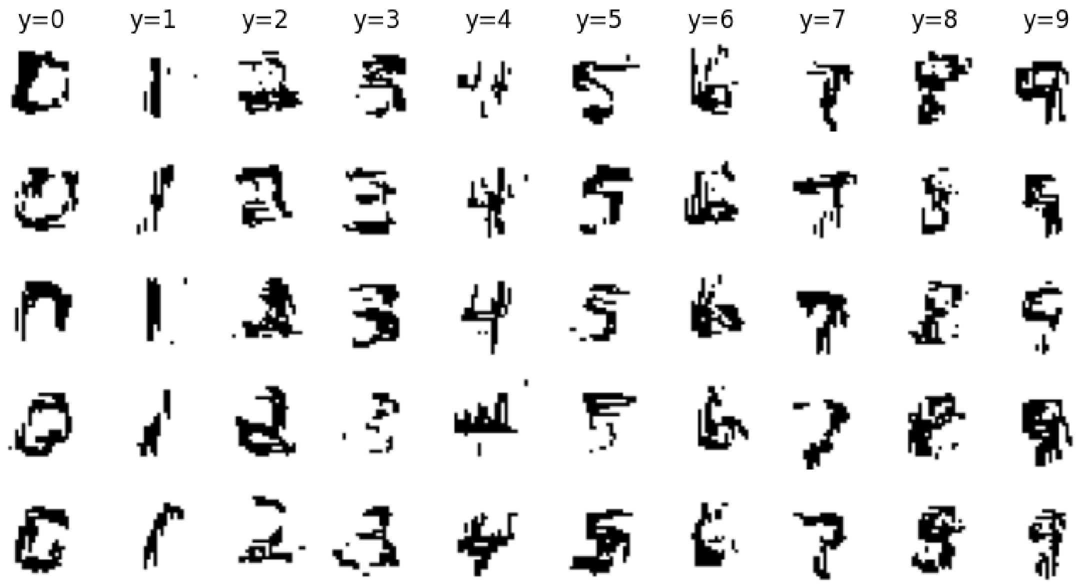
```
Error rate = 0.0728
Accuracy = 0.9272
```

4 Extra (not in the homework, I got bored)

Generating some images with the finished probabilistic model. This class of image generation is called autoregressive models.

[6]:
```
# for fun: autoregressive generation
def generate(y):
    image_padded = np.zeros([29,29])
    for i,j in itertools.product(list(range(28)), list(range(28))):
        parents_index = int(image_padded[i,j+1]*2+image_padded[i+1,j])
        p0 = probabilities[0,y,parents_index,i,j]
        p1 = probabilities[1,y,parents_index,i,j]
        n = p0+p1
        image_padded[i+1,j+1]=np.random.choice([0,1], p=(p0/n,p1/n))
    return image_padded[1:,1:]

# generate some examples
fig, ax = plt.subplots(5,10, figsize=(10,5))
plt.gray()
for i in range(10):
    ax[0,i].set_title(f"y={i}")
    for j in range(5):
        ax[j,i].imshow(1-generate(i))
        ax[j,i].axis('off')
```

```
plt.show()
```




```
[ ]:
```