CS561 HW05

October 9, 2023

1 Problem 1

Let $X \sim \mathcal{N}(\mu, \sigma^2)$

a) For any $a, b \in R$, $Y = aX + b$ is also Gaussian. Using the can see that the mean of Y is

$$\begin{aligned}\mathbb{E}[Y] &= \mathbb{E}[aX + b] \\ &= \int_{-\infty}^{\infty} (ax + b)\mathbb{P}(X = x)dx \\ &= a\int_{-\infty}^{\infty} x\mathbb{P}(X = x)dx + b\int_{-\infty}^{\infty} \mathbb{P}(X = x)dx \\ &= a\mu + b\end{aligned}$$

Moreover, we can compute the variance as such

$$\begin{aligned}\text{Var}(Y) &= \mathbb{E}[Y^2] - \mathbb{E}[Y]^2 \\ &= \mathbb{E}[a^2X^2 + 2abX + b^2] - (a\mu + b)^2 \\ &= a^2\mathbb{E}[X^2] + 2ab\mu + b^2 - a^2\mu^2 - 2ab\mu - b^2 \\ &= a^2(\mathbb{E}[X^2] - \mu^2) \\ &= a^2\sigma^2\end{aligned}$$

Therefore, $Y \sim \mathcal{N}(a\mu + b, a^2\sigma^2)$

b) Given $X \sim \mathcal{N}(\mu, \sigma^2)$, suppose $Y = aX + b \sim \mathcal{N}(0, 1)$. This means

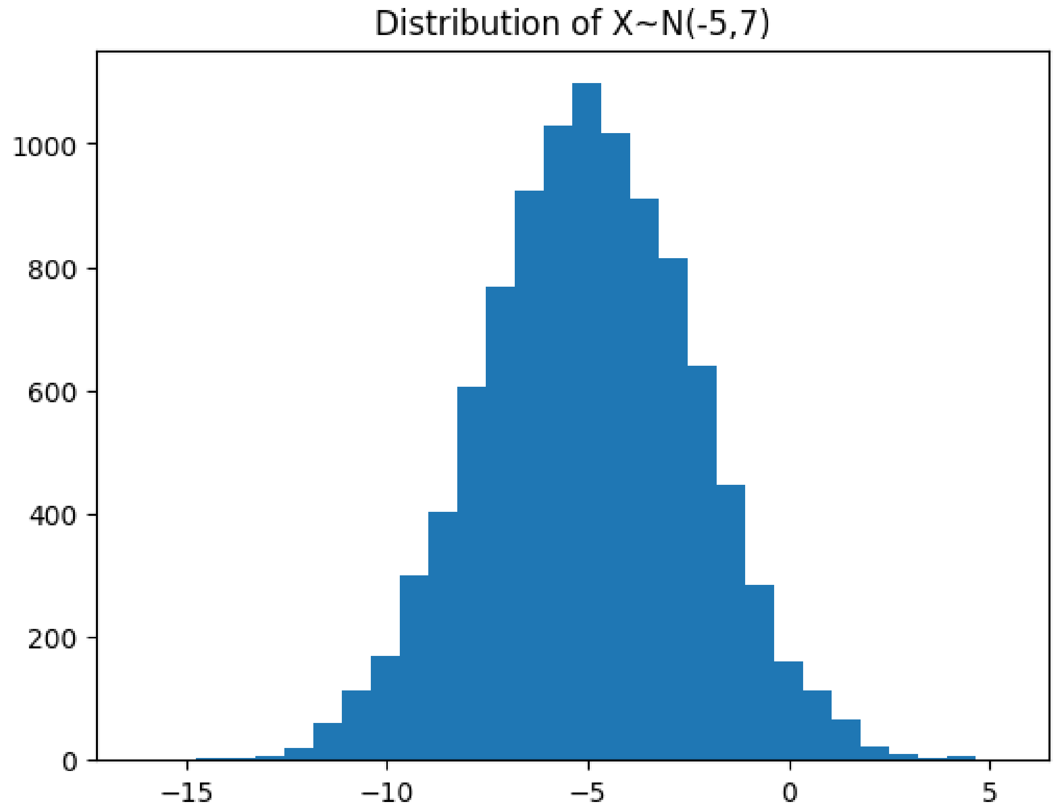
$$a\mu + b = 0 \text{ and } a^2\sigma^2 = 1$$

The second equation leads to $a = \frac{1}{\sigma}$, substituting in the first equation gives $b = -\frac{\mu}{\sigma}$

c) See the code below

[1]:
```python
# 1.c)
import numpy as np
import math
import matplotlib.pyplot as plt
samples = np.random.randn(10000) # scale by standard deviation
samples = samples * math.sqrt(7) # shift by mean
samples = samples - 5
plt.hist(samples, bins=30)
plt.title("Distribution of X~N(-5,7)")
```

```
plt.show()
```



2 Problem 2

Let $x \in \mathbb{R}^{n\times 1}$ be a random vector with mean $\mu_x \in \mathbb{R}^{n\times 1}$ and covariance matrix $\Sigma_x \in \mathbb{R}^{n\times n}$. Let $A \in \mathbb{R}^{m\times n}$ and $c \in \mathbb{R}^{n\times 1}$ be deterministic. Let $Y = A(x+c)$

a) Let $A = (a_{ij})_{i=1,j=i}^{n,m}$, then we can compute

$$\begin{aligned}
\mathbb{E}[Y] &= \mathbb{E}\left[Ax + Ac\right] \\
&= \mathbb{E}\left[Ax\right] + \mathbb{E}[Ac] \\
&= \mathbb{E}\left[\left(\sum_{i=1}^{m} a_{ij}x_i\right)_{j=1}^{m}\right] + Ac \\
&= \left(\sum_{i=1}^{m} a_{ij}\mu_{x_i}\right)_{j=1}^{m} + Ac \\
&= A\mu_x + Ac
\end{aligned}$$

b) Similarly, we can compute

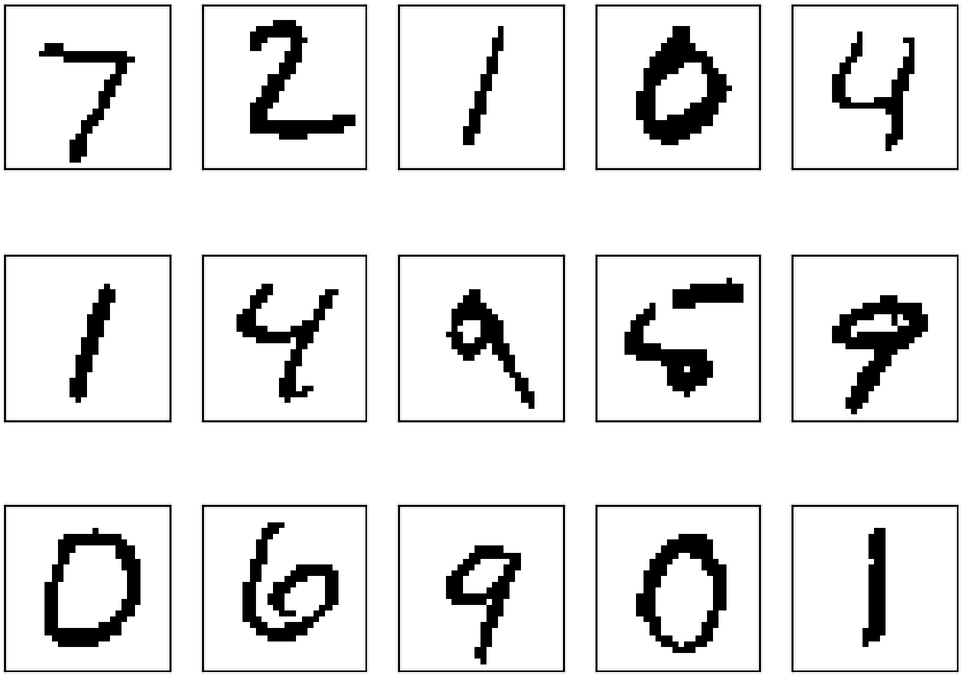
$$\begin{aligned}
\mathrm{Var}(Y) &= \mathbb{E}[(Y - \mathbb{E}[Y])(Y - \mathbb{E}[Y])^T] \\
&= \mathbb{E}[(Ax + Ac - A\mu_x - Ac)(Ax + Ac - A\mu_x - Ac)^T] \\
&= \mathbb{E}[(Ax - A\mu_x)(Ax - A\mu_x)^T] \\
&= \mathbb{E}[A(x - \mu_x)(x - \mu_x)^T A^T] \\
&= A\mathbb{E}[(x - \mu_x)(x - \mu_x)^T]A^T \\
&= A\Sigma_x A^T
\end{aligned}$$

3 Problem 3

[10]:

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from matplotlib import pyplot as plt
import warnings
warnings.filterwarnings("ignore")

(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.round(1.0*i/256) for i in x_train]
x_test_int = [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:
```



a) Assuming $p(x|y) = \prod_{i=1}^{n} p(x_i|y)$ where each x_i only takes in 0 or 1. Since each pixel is independent, we'd need to model each $p(x_i = 0|y)$ per y (we automatically can model $p(x_i|Y)$ immediately), meaning there are $n = 784 \times 10 = 7840$ values to model.

b) It is not reasonable to assume each pixel value is independent for images for two reasons: image can be shifted left and right (but still refers to the same class) and images have heirarchial and spatial relation (i.e. patterns) which would not be considered.

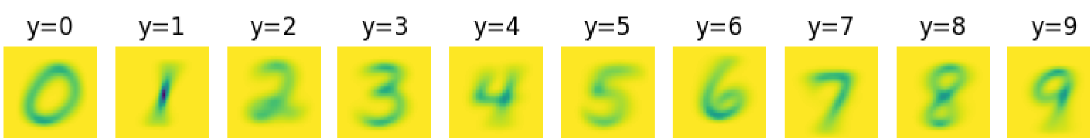
c-h) See the code below

[3]:
```
# 3.c)
x_train_flatten = np.array(x_train_int).reshape(-1,784)
# precompute probabilities to mitigate computational burden
# the value at index [y,i,c] refers to p(xi=c|y=y)
#shape = classes x pixels x binary
log_p_xi_given_y = np.zeros([10,784,2])
eps = 0.7
for y in range(10):
    all_x = x_train_flatten[y_train==y]
    for i in range(784):
        all_xi = all_x[:,i]
        for c in [0,1]:
            count = np.sum(all_xi==c)
            log_p_xi_given_y[y,i,c] = np.log((count+eps)/(len(all_xi)+eps))

# store in 10x28x28 array separately for 0 and 1
```

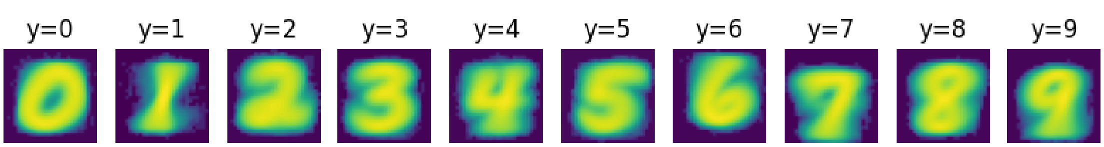
```
log_prob_0 = log_p_xi_given_y[:,:,0].reshape(10,28,28)
log_prob_1 = log_p_xi_given_y[:,:,1].reshape(10,28,28)
```

[4]:
```
# imshow "model" images for each class
print("For pixel value = 0")
fig, ax = plt.subplots(1,10, figsize=(10,1))
for i in range(10):
    ax[i].imshow(log_prob_0[i], vmin=np.min(log_prob_0), vmax=np.
 ↪max(log_prob_0))
    ax[i].axis('off')
    ax[i].set_title(f"y={i}")
plt.show()
print("For pixel value = 1")
fig, ax = plt.subplots(1,10, figsize=(10,1))
for i in range(10):
    ax[i].imshow(log_prob_1[i], vmin=np.min(log_prob_1), vmax=np.
 ↪max(log_prob_1))
    ax[i].axis('off')
    ax[i].set_title(f"y={i}")
plt.show()
```

For pixel value = 0



For pixel value = 1



[5]:
```
# 3.d)
x_test_arr = np.array(x_test_int)
def prob(im): #sum of log prob for all pixels
    prob0 = (log_prob_0 * (1-im)).sum(-1).sum(-1)
    prob1 = (log_prob_1 * im).sum(-1).sum(-1)
    return prob0 + prob1
y_test_pred = np.array([np.argmax(prob(im)) for im in x_test_arr])
```

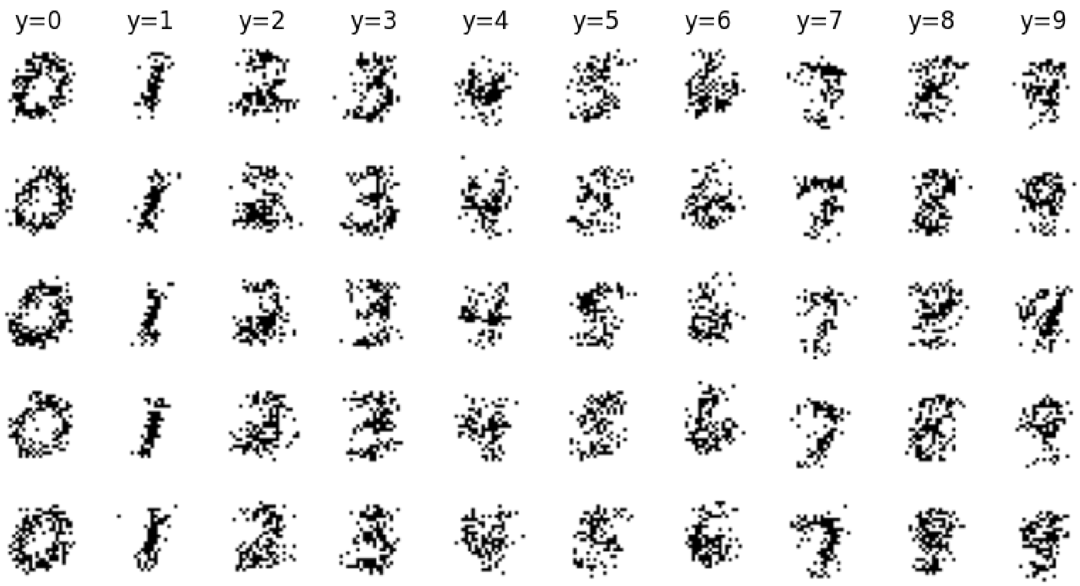
[6]:
```
# 3.e)
print(f"Error rate = {np.sum(y_test_pred!=y_test)/len(y_test)}")
print(f"Accuracy = {np.sum(y_test_pred==y_test)/len(y_test)}")
```

```
Error rate = 0.1565
Accuracy = 0.8435
```

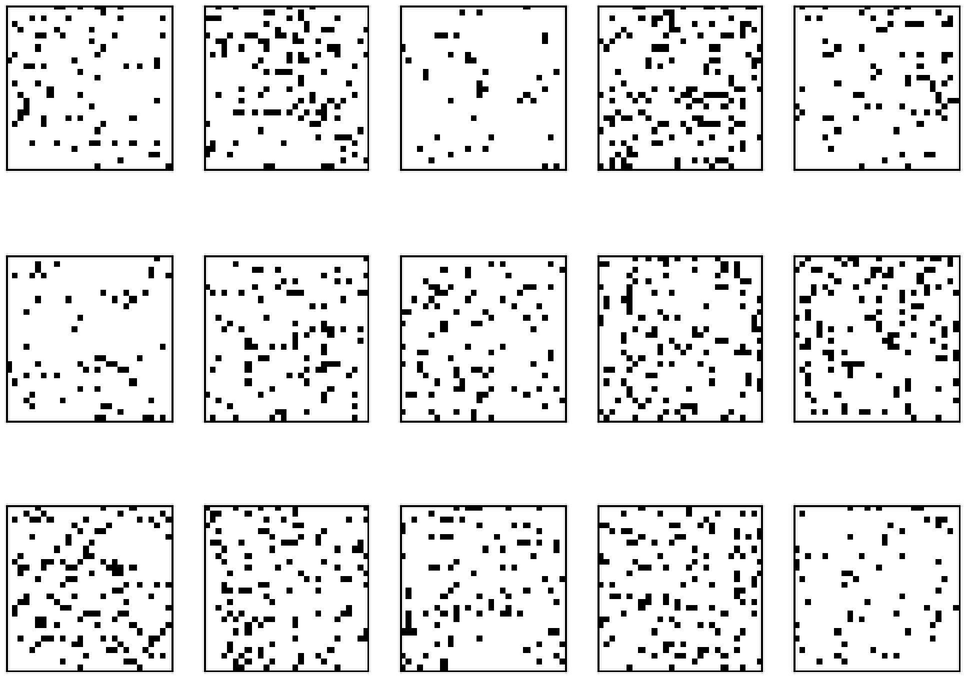
[7]:
```
# 3.f)
# log frequencies
py = np.log(np.array([np.sum(y_train==i)/len(y_train) for i in range(10)]))
x_test_arr = np.array(x_test_int)
def prob_with_py(im): #sum of log prob for all pixels
    prob0 = (log_prob_0 * (1-im)).sum(-1).sum(-1)
    prob1 = (log_prob_1 * im).sum(-1).sum(-1)
    return prob0 + prob1 + py
y_test_pred_with_py = np.array([np.argmax(prob_with_py(im)) for im in␣
 ↪x_test_arr])
print(f"Error rate = {np.sum(y_test_pred_with_py!=y_test)/len(y_test)}")
print(f"Accuracy = {np.sum(y_test_pred_with_py==y_test)/len(y_test)}")
```

```
Error rate = 0.1565
Accuracy = 0.8435
```

[8]:
```
# 3.g)
# generate an image given y
def generate(y):
    prob0 = np.exp(log_prob_0[y].reshape(784))
    prob1 = np.exp(log_prob_1[y].reshape(784))
    norm = prob0+prob1
    return np.array([np.random.choice([0,1], p=(prob0[i]/norm[i], prob1[i]/
 ↪norm[i])) for i in range(784)]).reshape(28,28)
# 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()
```



[9]:
```
# 3.h)
x_test_permuted = np.array([np.random.permutation(im.reshape(-1)).
 ↪reshape(28,28) for im in x_test_int])
show_images(x_test_permuted, 3, 5)
x_test_arr_permuted = np.array(x_test_permuted)
def prob(im): #sum of log prob for all pixels
    prob0 = (log_prob_0 * (1-im)).sum(-1).sum(-1)
    prob1 = (log_prob_1 * im).sum(-1).sum(-1)
    return prob0 + prob1
y_test_pred_permuted = np.array([np.argmax(prob(im)) for im in␣
 ↪x_test_arr_permuted])
print(f"Error rate = {np.sum(y_test_pred_permuted!=y_test)/len(y_test)}")
print(f"Accuracy = {np.sum(y_test_pred_permuted==y_test)/len(y_test)}")
```



```
Error rate = 0.895
Accuracy = 0.105
```

We can see that the error rate increased drastically once we permute the pixels. This means our classifier considers positional information in its classification.