Week 1

HW 1, Part 1

By James Camacho

Part 1.1, Mixture of Bernoullis

1. The probability we get x_d given p_d is $x_d p_d + (1 - x_d)(1 - p_d)$, i.e. p_d if x_d is one, otherwise $1 - p_d$. Note we can sum (as opposed to using cases) as one of the terms is always zero. We have $P(x|p)$ is the product of all these, or

$$P(x|p) = \prod_{d=1}^{D} x_d p_d + (1 - x_d)(1 - p_d).$$

2. Basically by definition,

$$P(x^{(i)}|\mathbf{p}, \pi) = \sum_{k=1}^{K} \pi(k) P(x^{(i)}|p^{(k)})$$

3. It's the product for each $x^{(i)} \in X$, so taking logs we get

$$\log P(X|\mathbf{p}, \pi) = \sum_{i=1}^{n} \log P(x^{(i)}|\mathbf{p}, \pi)$$

Part 1.2, Expectation Step

4. Each of the k elements in $z^{(i)}$ are chosen based on the distribution π. So

$$P(z^{(i)}|\pi) = \prod_{k=1}^{K} z_k^{(i)} \pi(k) + \left(1 - z_k^{(i)}\right)(1 - \pi(k)).$$

We have

$$P(x^{(i)}|z^{(i)}, \mathbf{p}, \pi) = \sum_{k=1}^{K} P(x^{(i)}|p^{(k)}) z_k^{(i)}$$

(or alternatively $\prod_{k=1}^{K} P(x^{(i)}|p^{(k)})^{z_k^{(i)}}$). Note that π doesn't play a role because we are already given which Bernoulli distribution x is drawn from with the indicator vector $z^{(i)}$.

5. Note that

$$P(x^{(i)}, z^{(i)}|\mathbf{p}, \pi) = P(z^{(i)}|\pi) \cdot P(x^{(i)}|z^{(i)}, \mathbf{p}, \pi).$$

So $P(Z, X|\pi, \mathbf{p})$ will be the product for each i, i.e.

$$\prod_{i=1}^{n} P(z^{(i)}|\pi) \cdot P(x^{(i)}|z^{(i)}, \mathbf{p}, \pi)$$

6. I'm assuming $\pi_k = \pi(k)$. So, remember how I used

$$x_d p_d + (1 - x_d)(1 - p_d)$$

in 1.1.1? Well it's the exact same value as

$$p_d^{x_d}(1 - p_d)^{1-x_d}.$$

Now, $E[z_k^{(i)}|x^{(i)}, \pi, \mathbf{p}]$ is the probability $p^{(k)}$ was chosen (π_k) times the probability we get $x^{(i)}$ given $p^{(k)}$ ($P(x^{(i)}|p^{(k)})$, see 1.1.1) divided by the probability we get $x^{(i)}$ (see 1.1.2). This works out to the given expression for η:

$$\eta(z_k^{(i)}) = \frac{\pi_k \prod_{d=1}^{D} (p_d^{(k)})^{(x_d^{(i)})}(1 - p_d^{(k)})^{1-x_d^{(i)}}}{\sum_j \pi_j \prod_{d=1}^{D} (p_d^{(j)})^{(x_d^{(i)})}(1 - p_d^{(j)})^{1-x_d^{(i)}}}$$

Now,

$$\begin{aligned} \log P(Z, X|\tilde{\mathbf{p}}, \tilde{\pi}) &= \sum_{i=1}^{N} \log P(z^{(i)}|\tilde{\pi}) + \log P(x^{(i)}|z^{(i)}, \tilde{\mathbf{p}}, \tilde{\pi}) \\ &= \sum_{i=1}^{N} \sum_{k=1}^{K} \left[\log \tilde{\pi}_k + \sum_{d=1}^{D} \left(x_d^{(i)} \log \tilde{p}_d^{(k)} + (1 - x_d^{(i)}) \log(1 - \tilde{p}_d^{(k)}) \right) \right] \end{aligned}$$

To find that big expected value, we have to weight the summation by the chance we actually get the summand in question, or $\eta(z_k^{(i)})$. This gives us the desired:

$$E[\log P(Z, X|\tilde{\mathbf{p}}, \tilde{\pi}) \mid X, \mathbf{p}, \pi] = \sum_{i=1}^{N} \sum_{k=1}^{K} \eta(z_k^{(i)}) \left[\log \tilde{\pi}_k + \sum_{d=1}^{D} \left(x_d^{(i)} \log \tilde{p}_d^{(k)} + (1 - x_d^{(i)}) \log(1 - \tilde{p}_d^{(k)}) \right) \right]$$

Part 1.3, Maximization step

1. Taking a gradient with respect to $\tilde{p}^{(k)}$ gives

$$\sum_{i=1}^{N} \eta(z_k^{(i)}) \left[\frac{x^{(i)}}{\tilde{p}^{(k)}} - \frac{(1 - x^{(i)})}{1 - \tilde{p}^{(k)}} \right]$$

which we want equal to zero. Clearing denominators and a little algebra gives

$$\underset{\sim}{p}^{(k)} = \frac{\sum_{i=1}^{N} \eta(z_k^{(i)}) x^{(i)}}{\sum_{i=1}^{N} \eta(z_k^{(i)})}.$$

2. Taking a gradient with respect to $\tilde{\pi}_k$ gives

$$\frac{1}{\tilde{\pi}_k} \sum_{i=1}^{N} \eta(z_k^{(i)}).$$

We have the constraint $\sum_k \tilde{\pi}_k = 1$, and taking a gradient of that gives 1. So Lagrange multipliers tells us

$$\frac{1}{\tilde{\pi}_k} \sum_{i=1}^{N} \eta(z_k^{(i)}) = \lambda$$

for a constant λ, which we can solve to be $1/\sum_{k'} N_{k'}$. This gives the desired

$$\tilde{\pi}_k = \frac{N_k}{\sum_{k'} N_{k'}}.$$

Note: $N_k = \sum_{i=1}^{N} \eta(z_k^{(i)})$.

HW 1, Part 2

By James Camacho

Our goal is to implement a expectation-maximization clustering algorithm. We will make use of the following variables:

```
Xs      N x D (N vertices of dimension D)
p       K x D (K clusters of dimension D)
mix_p   K x 1 (chance of being in each cluster, aka pi)
eta     N x K (eta[i, k] = eta(z_k^i))
Copy
```

```
import numpy as np
from pandas import read_csv
import matplotlib.pyplot as plt
from tqdm.notebook import tqdm
#from tqdm import tqdm
Copy
```

```
# 2.1.1
def eta(Xs, p, mix_p):
    """
    Returns eta.
    """
    h = Xs @ np.log(p).T + (1-Xs) @ np.log(1-p).T
    h += np.log(mix_p)[None,:]
    h -= np.mean(h) # Normalizes h so no overflow errors.
    h = np.exp(h)
    h /= np.sum(h,axis=1)[:,None]
    return h

# 2.1.2
def maximize(eta, Xs, alpha1=1e-8, alpha2=1e-8):
    """
    Returns maximum values for p, mix_p.
    alpha1 and alpha2 are smoothing parameters.
    """
    p_num = eta.T @ Xs + alpha1
    p_denom = np.sum(eta, axis=0)[:,None] + alpha1 * Xs.shape[-1] # i.e. alpha1 * D

    mix_p_num = np.sum(eta, axis=0) + alpha2
    mix_p_denom = np.sum(eta) + alpha2 * eta.shape[-1] # i.e. alpha2 * K

    return p_num / p_denom, mix_p_num / mix_p_denom


# 2.1.3
def cluster(Xs, K, iters=20):
    """
    Determines K clusters for the Xs.
    """
    N, D = Xs.shape
    p = np.random.random((D,K))
    p /= np.sqrt(np.sum(p**2, axis=0))[None,:]
    p = p.T
    mix_p = np.ones(K) / K
    for i in tqdm(range(iters), desc="Clustering..."):
        h = eta(Xs, p, mix_p)
        p, mix_p = maximize(h, Xs)

    return p, mix_p

# 2.1.3
def label(Xs, p, mix_p):
    """
    Finds which cluster each X belongs to.
    """
    h = eta(Xs, p, mix_p)
    return np.argmax(h, axis=1)

def group(data, labels):
    """
    Groups the Xs by the labels.
    """
    groups = [[] for i in range(max(labels) + 1)]
    for i, l in enumerate(labels):
        groups[l].append(data[i])
    return groups
Copy
```

Okay great! We got the algorithm all sorted out. Let's try it with 500 images from the MNIST dataset:

```
IMAGES = 500
K = 20 # Number of clusters.

# Prepare image data
if IMAGES:
    mnist = next(read_csv("mnist.csv", chunksize=IMAGES))
else:
    mnist = read_csv("mnist.csv")

digits = mnist["label"].to_numpy() # For use later.
mnist.drop("label", 1, inplace=True)

# Can round for binary x values, but since we have grayscale we might as well use it.
Xs = mnist.to_numpy()/255

# Find the p, mix_p for the clusters.
p, mix_p = cluster(Xs, K)

# Graph
rows = int(K**0.5)
cols = (K + rows - 1) // rows
fig, axes = plt.subplots(rows, cols)
for i in range(rows):
    for j in range(cols):
        im = np.reshape(p[i*rows + j], (28, 28))
        ax = axes[i, j] if rows > 1 else axes[j]
        ax.imshow(im, cmap='gray')
plt.show()
Copy
```

```
labels = label(Xs, p, mix_p)
groups = group(digits, labels)

cluster_digits = np.array([len(set(g)) for g in groups])

print(f"Digits per cluster:\n{cluster_digits}")
print(f"Mean digits per cluster: {np.mean(cluster_digits)}")
print(f"Single-digit clusters: {np.where(cluster_digits == 1)[0]}")
Copy
```

```
Digits per cluster:
[4 4 3 2 6 1 8 5 7 2 1 7 5 3 2 4 1 6 2 3]
Mean digits per cluster: 3.8
Single-digit clusters: [ 5 10 16]
Copy
```

Most clusters have more than one unique digit, but some do pick out exactly one digit.

HW 1, Part 3

By James Camacho

3.1, kernel PCA

1. From its definition,

$$C = E[x^2] - E[x]^2 = \frac{1}{N}\sum_{i=1}^{N} \phi(x_i)\phi(x_i)^T - \left(\overline{\phi(x)}\right)\left(\overline{\phi(x)}\right)^T,$$

   where $\overline{\phi(x)} = \frac{1}{N}\sum_{i=1}^{N} \phi(x_i)$. An equivalent formula is

$$C = E[(x - \overline{\phi(x)})^2] = \frac{1}{N}\sum_{i=1}^{N} (\phi(x_i) - \overline{\phi(x)})(\phi(x_i) - \overline{\phi(x)})^T.$$

   This latter one will be better for the next part.

2. As each column of C is a linear combination of the vectors $(\phi(x_i) - \overline{\phi(x)})$, then Cv can only produce linear combinations of those vectors. So, λv is a linear combination of them, implying v is a linear combination of them, i.e.

$$v = \sum_{i=1}^{N} \alpha_i \left(\phi(x_i) - \overline{\phi(x)}\right).$$

3. We have

$$\begin{aligned}(\widetilde{K}\alpha)_i &= \sum_{j=1}^N \left\langle \phi(x_i) - \overline{\phi(x)}, \alpha_j(\phi(x_j) - \overline{\phi(x)})\right\rangle \\ &= \left\langle \phi(x_i) - \overline{\phi(x)}, v\right\rangle\end{aligned}$$

by putting the summand inside the inner product. Now,

$$\begin{aligned}(\widetilde{K}(\widetilde{K}\alpha))_i &= \sum_{j=1}^N \left\langle \phi(x_i) - \overline{\phi(x)}, (\phi(x_j) - \overline{\phi(x)})\left\langle \phi(x_j) - \overline{\phi(x)}, v\right\rangle\right\rangle \\ &= \left\langle \phi(x_i) - \overline{\phi(x)}, \sum_{j=1}^N (\phi(x_j) - \overline{\phi(x)})(\phi(x_j) - \overline{\phi(x)})^T v\right\rangle \\ &= \left\langle \phi(x_i) - \overline{\phi(x)}, NCv\right\rangle \\ &= \left\langle \phi(x_i) - \overline{\phi(x)}, N\lambda v\right\rangle.\end{aligned}$$

Pulling out the $N\lambda$ gives the desired.

4. Left multiplying by $\widetilde{K}$ gives the desired.
5. Looking at a particular element, we have

$$\begin{aligned}\widetilde{K}_{ij} &= \langle \phi(x_i), \phi(x_j)\rangle - \left\langle \phi(x_i), \overline{\phi(x)}\right\rangle - \left\langle \phi(x_j), \overline{\phi(x)}\right\rangle + \left\langle \overline{\phi(x)}, \overline{\phi(x)}\right\rangle \\ &= K_{ij} - (Kee^T)_{ij} - (ee^TK)_{ij} + (ee^TKee^T)_{i,j} \\ &= (I - ee^T)K(I - ee^T).\end{aligned}$$

6. The square of the norm of v would be

$$\begin{aligned}\left\langle \sum_{i=1}^N \alpha_i(\phi(x_i) - \overline{\phi(x)}), \sum_{j=1}^N \alpha_j(\phi(x_j) - \overline{\phi(x)})\right\rangle &= \sum_{i=1}^N \alpha_i \sum_{j=1}^N \left\langle \phi(x_i) - \overline{\phi(x)}, \phi(x_j) - \overline{\phi(x)}\right\rangle \alpha_j \\ &= \alpha^T\widetilde{K}\alpha.\end{aligned}$$

So we need to multiply it by $\frac{1}{\sqrt{\alpha^T\widetilde{K}\alpha}}$.

7. Let $V = [v_1\ v_2\ v_3\ \ldots\ v_d]$. The new coordinates for x would be $V^{-1}x$. We can calculate V without ever calculating $\phi(x)$ as it only involves inner products, so we can calculate $V^{-1}x$ without ever explicitly calculating $\phi(x)$.

HW 1, Part 4

4.1, Huber Function

1. Let $f(\lambda) = \lambda + \frac{x^2}{\lambda + d}$, so

$$f'(\lambda) = 1 - \frac{x^2}{(\lambda + d)^2}.$$

If $|x| \le d$ then $\frac{x^2}{(\lambda+d)^2} \le 1$ and f is monotonically increasing on $[0, \infty)$. So the minimum occurs when $\lambda = 0$, which gives $B(x, d) = \frac{x^2}{d}$. Otherwise, the minimum occurs when $f'(\lambda) = 0$, which directly solving gives $\lambda = |x| - d$, and $B(x, d) = 2|x| - d$.

2. Yes, B is convex in x. We have $\frac{\partial^2}{\partial x^2} B = 2$ for $|x| \le d$ and 0 for $|x| > d$, which is nonnegative, enough to prove convexity.
3. We have
$$\nabla B = \begin{cases} (\frac{2x}{d}, -\frac{x^2}{d}) & |x| \le d \\ (\frac{2x}{|x|}, -1) & |x| > d. \end{cases}$$
4. When minimizing you go in the opposite direction of the gradient. As d is a constant, this results in a pull of $-\frac{2x}{\max(|x|,d)}$. As d decreases, x will be pulled more strongly towards 0.

4.2, An application of using Huber loss to solve dual problem.
Consider the optimization problem

$$\min_x \left[f(x) := \sum_{i=1}^N \tfrac{1}{2} d_i x_i^2 + r_i x_i \right]$$
$$\text{s.t.} \quad a^T x = 1, \quad x_i \in [-1, 1] \ \text{ for } \ i = 1, 2, \dots, n.$$

1. If $||a||_1 \le 1$ then we only get $a^T x = 1$ for all $|x_i| \ge 1$, so it's not strictly feasible. Now, if $||a||_1 > 1$, we can use a greedy algorithm to pick the x's and get a strictly feasible solution.
2. Note: Problem 4.2.4 switches the standard convention of μ and λ so I have reflected that here. The Lagrangian is

$$L(x, \mu, \lambda) = \sum_{i=1}^N \left(\frac{1}{2} d_i x_i^2 + r_i x_i \right) + \mu(a^T x - 1) + \sum_{i=1}^N \frac{1}{2} \lambda_i \left(x_i^2 - 1\right)$$

Taking a gradient with respect to x and setting it to zero gives

$$x_i^* = -\frac{r_i + \mu a_i}{d_i + \lambda_i}.$$

Plugging back in gives the dual problem to maximize

$$q(\mu, \lambda) = -\mu - \frac{1}{2} \sum_{i=1}^N \frac{(r_i + \mu a_i)^2}{d_i + \lambda_i}.$$

subject to $\lambda_i \ge 0$.
3. According to the KKT conditions, an optimal solution x^* must satisfy:
- **Stationarity**: $d_i x_i^* + r_i + \mu a_i + \lambda_i x_i^* = 0$
- **Primal Feasibility**: $a^T x^* - 1 = 0$, and $(x_i^*)^2 - 1 \le 0$ for $1 \le i \le N$.
- **Dual Feasibility:** $\lambda_i \ge 0$.
- **Complementary Slackness:** $\sum_{i=1}^N \lambda_i((x_i^*)^2 - 1) = 0$.

The first and third KKT conditions were how we obtained the dual problem, and primal feasibility is basically Slater's condition and shown in 4.2.1 above. The final condition, complementary slackness, finishes characterizing the optimal solution, as Lagrange multipliers could be used to find λ given μ, and then you just need to maximize based on μ.

4. Our problem is equivalent to

$$\min_{\mu} \mu + \sum_{i=1}^{N} \frac{1}{2} \min_{\lambda_i} \frac{(r_i + \mu a_i)^2}{d_i + \lambda_i}$$

From the slack condition, we have the following two cases for that inner minimum:

$$\lambda_i = 0 \implies \min_{\lambda_i} \frac{(r_i + \mu a_i)^2}{d_i + \lambda_i} = \frac{(r_i + \mu a_i)^2}{d_i},$$

or

$$\lambda_i \neq 0 \implies |x_i^*| = 1 \Longleftrightarrow \lambda_i = |r + \mu a_i| - d_i$$

which gives a minimum of

$$\min_{\lambda_i} \frac{(r_i + \mu a_i)^2}{d_i + \lambda_i} = |r_i + \mu a_i| < \frac{(r_i + \mu a_i)^2}{d_i} \qquad \text{if } |r_i + \mu a_i| > d_i.$$

The minimum always occurs in the same place as the Huber function, even if the minimum isn't exactly the same, so the problem is equivalent to finding

$$\min_{\mu} \mu + \frac{1}{2} \sum_{i=1}^{N} B(r_i + \mu a_i, d_i).$$

5. We have

$$-\frac{1}{2} \cdot \frac{\partial B(r_i + \mu a_i, d_i)}{\partial \mu} = a_i \cdot \begin{cases} (r_i + \mu a_i)/d_i & |r_i + \mu a_i| \leq d_i \\ \operatorname{sgn}(r_i + \mu a_i) & |r_i + \mu a_i| > d_i. \end{cases}$$

where $\operatorname{sgn}(x) = |x|/x$ is the sign of x. The sum of all of these needs to equal 1. We cut the real line at the $4N$ points where $|r_i + \mu a_i| = d_i$ or $|r_i + \mu a_i| = 0$. It takes $O(N)$ time to find the cuts, but we need to sort them so we end up with $O(N \log N)$ time complexity here. Now, we'll have a bunch of intervals: $I_1, I_2, I_3, \ldots I_{4N+1}$, some of which may be empty. For each of these intervals I_k, we create latent variables

$$z_i = \begin{cases} 1 & |r_i + \mu a_i| \leq d_i \\ 0 & |r_i + \mu a_i| > d_i \end{cases},$$

$$s_i = \operatorname{sgn}(r_i + \mu a_i),$$

The optimal μ that falls in this interval would be

$$\mu_k = \frac{1 - \sum_{i=1}^{N} [z_i a_i r_i / d_i + (1 - z_i) a_i s_i]}{\sum_{i=1}^{N} z_i a_i^2 / d_i}.$$

It would be a waste of time to constantly evaluate those sums (not to mention the latent variables), so we keep track of the numerator and denominator separately. As we loop through the cutting points, we only need to change the terms within each sum associated with that particular point, a constant time operation. So we can compute all the μ_k's in linear time. We ignore any $\mu_k \notin I_k$ and sort the rest. Worst case this is $O(N \log N)$. Finally, we need to find the best μ from these. It takes $O(N)$ time to plug μ into the minimizer function, but luckily the function is convex. We can use a binary search on our sorted list of μ_k's in $O(N \log N)$ time. The overall time complexity is $O(N \log N)$.

6. Once you have μ, you plug it into each of the i Huber functions. It takes constant time to find the best λ_i for each one. Once μ and λ are determined, we can use the formula from above:

$$x_i^* = -\frac{r_i + \mu a_i}{d_i + \lambda_i}.$$