Q1

February 18, 2021

1 Question 1

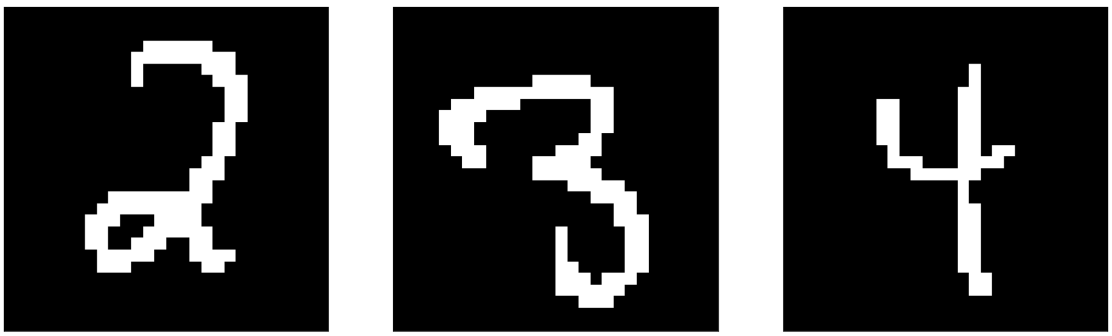
See Chapter 9, Section 9.3.3 in Pattern Recognition and Machine Learning book by Christopher Bishop for a mixture of Bernoulli distributions. Use EM algorithm to obtain results as in Figure 9.10. Run K-Means algorithm on same dataset and compare the results.

[1]:
```python
import numpy as np
import pandas as pd
import time
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_openml
import matplotlib.pyplot as plt
from PIL import Image
np.random.seed(42)

class Config:
    N = 600
    K = 3
    D = 784
    iterations = 50
```

[2]:
```python
def load_mnist(toBinary=True):
    mnist = pd.read_csv('mnist_train.csv').values
    X_train, y_train = mnist[:, 1:], mnist[:, 0]

    # take only digits 2, 3 and 4
    take = (y_train >= 2) & (y_train <= 4)
    X_train, y_train = X_train[take], y_train[take]


    # making arrays have only zeros or ones
    if toBinary:
        X_train = ((X_train/255.0) > .5).astype(int)

    # make sure you have a total of 600 examples
    train_size = Config.N / y_train.shape[0]
    X_train, _, y_train, _ = train_test_split(X_train, y_train,␣
↪train_size=train_size, stratify=y_train, random_state=42)
```

```python
    # show one example from each class
    fig, axs = plt.subplots(1, 3, figsize=(18, 6))
    for i, c in enumerate([2, 3, 4]):
        img = X_train[y_train == c][0].reshape((28, 28))
        axs[i].imshow(img, cmap='CMRmap')
        axs[i].set(xticks = [], yticks=[])
    plt.show()

    for c in [2, 3, 4]:
        t = (y_train==c).sum()
        print(f"Class={c}, {t} examples")
    print(f"Total={Config.N} examples")
    return X_train, y_train
```

[3]:
```python
X_train, y_train = load_mnist()
```



```
Class=2, 199 examples
Class=3, 205 examples
Class=4, 196 examples
Total=600 examples
```

1.1 EM algorithm for Multivariate Bernoulli Mixture

One iteration of EM updates π_k and μ_k as follows

$P(x|\mu_k) = \prod_{i=1}^{D} \mu_{ki}^{x_i}(1-\mu_{ki})^{(1-x_i)}$

$\gamma_{nk} = \frac{\pi_k P(x_n|\mu_k)}{\sum_{j=1}^{K} \pi_j P(x_n|\mu_j)}$

$N_k = \sum_{n=1}^{N} \gamma_{nk}$

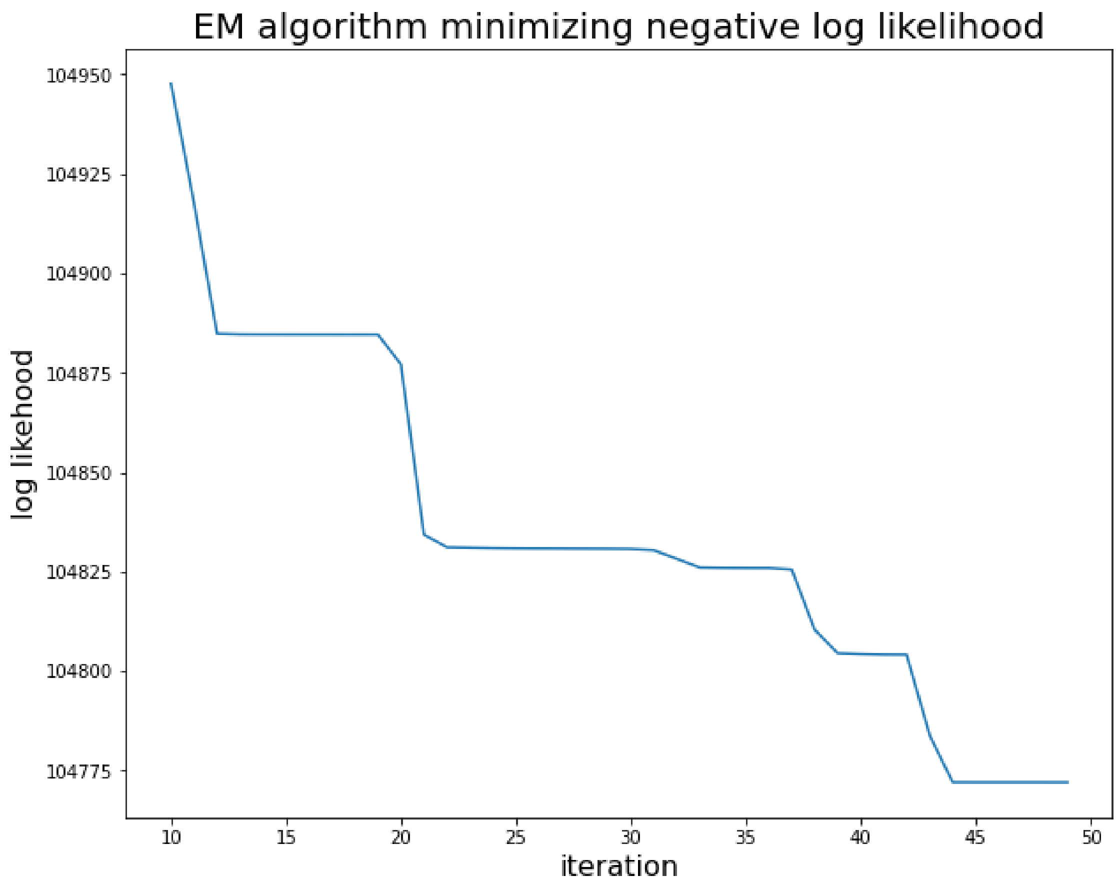
$\mu_k = \frac{\sum_{n=1}^{N} \gamma_{nk} x_n}{N_k}$

[4]:
```python
class Params:
    pi = np.ones((Config.K,1)) / Config.K                                          # pi_k
```

```python
    mu = np.random.uniform(low=.25, high=.75, size=(Config.K, Config.D)) # mu_kj
    mu /= mu.sum(axis=1, keepdims=True)           # sum mu_kj over j should be 1

# helpers
def prob(X_nd, k): # examples, k -> [P(example | k)]
    mu_1d = Params.mu[k:k+1]
    p_nd = mu_1d**X_nd * (1- mu_1d)**(1-X_nd) #(1, d)*(n, d)
    p_n = np.prod(p_nd, axis=1) # product over dimensions, axis=1
    return p_n

def getGamma(X_nd):
    g_kn = np.array([
        Params.pi[k, 0] * prob(X_nd, k)
        for k in range(Config.K)
    ]) # (k n)
    denom = g_kn.sum(axis=0, keepdims=True)
    denom[denom == 0.0] = 1
    g_kn /= denom # sum over clusters (k) must be one

    return g_kn, denom
```

[5]:
```python
def em_one_iteration(X_nd):
    """Performs one iteration of the EM algorithm for bernoulli mix model
    Args
        * X_nd is an np.array with n examples, each of d dimensions
    Function
        * updates Params.mu and Params.pi
    """
    g_kn, denom = getGamma(X_nd)

    # N_k is gamma summed over n
    N_k = g_kn.sum(axis=1, keepdims=True) # shape (k,)

    # new mu is examples averaged over gamma
    mu_kd = np.matmul(g_kn, X_nd)
    Params.mu = mu_kd / N_k
    Params.pi = N_k / Config.N

    # return log likelihood
    neg_log_likelihood = - np.log(denom).sum()
    return neg_log_likelihood

neg_log_likelihoods = [
    em_one_iteration(X_train)
    for i in range(Config.iterations)
]
```

[6]:
```
fig, ax = plt.subplots(figsize=(10, 8))
ax.plot(range(10, Config.iterations), neg_log_likelihoods[10:])
ax.set_ylabel('log likehood', fontsize=16)
ax.set_xlabel('iteration', fontsize=16)
ax.set_title('EM algorithm minimizing negative log likelihood', fontsize=20);
```



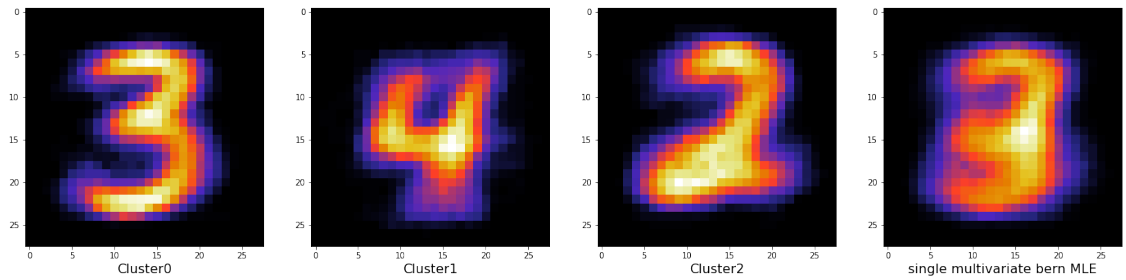
1.2 Most likely estimate for Single Multivariate Bernoulli

$Z = [z_i] \in R^d$

$Z_{mle} = \frac{\sum_{n=1}^{N} X_i}{N}$

[7]:
```
# display EM algo mus
images = (Params.mu * 255).reshape((Config.K, 28, 28)).astype(np.uint8)
fig, axs = plt.subplots(1, 4, figsize=(24, 6))
for k in range(Config.K):
    axs[k].imshow(images[k], cmap='CMRmap')
    axs[k].set_xlabel(f'Cluster{k}', fontsize=16)

img = (X_train.mean(axis=0) * 255).reshape((28, 28)).astype(np.uint8)
```

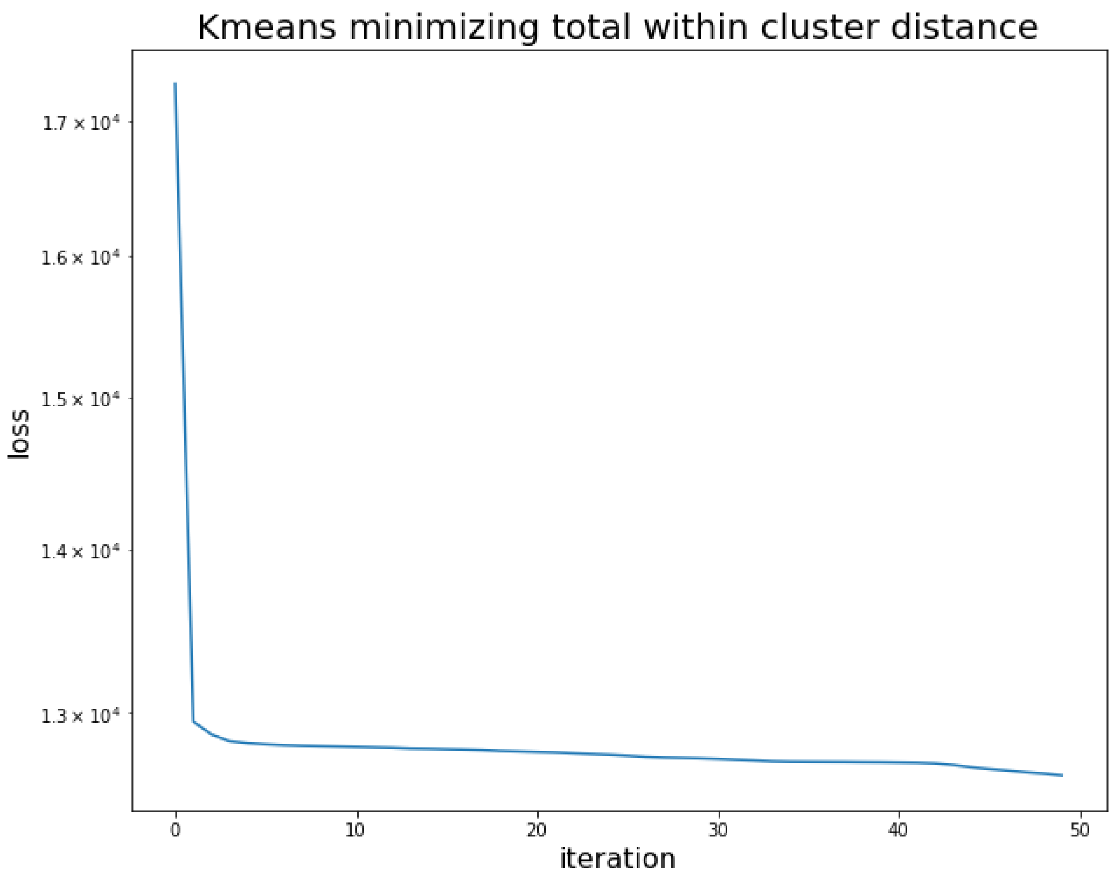
```
axs[Config.K].imshow(img, cmap='CMRmap')
axs[Config.K].set_xlabel('single multivariate bern MLE', fontsize=16)
plt.show()
```

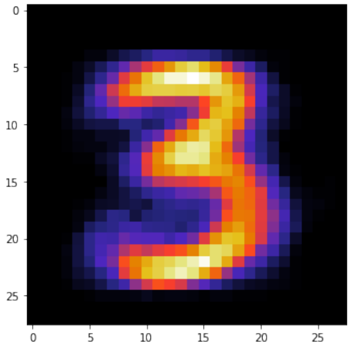


1.3 K-means clustering for mnist

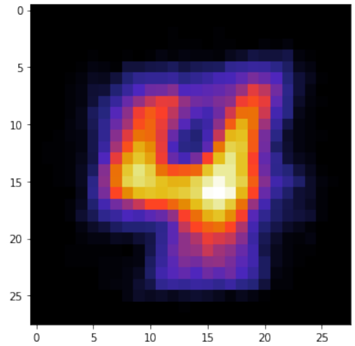
[8]:
```
# standardize data
sigma = X_train.std(axis=0, keepdims=True)
sigma[sigma == 0] = 1
X_stan = X_train  / sigma

# init assign
z_n  = np.random.choice(np.arange(Config.K), Config.N)

# init means as K random points from data
idx  = np.random.choice(np.arange(Config.N), Config.K)
Params.mu = X_stan[idx] * 1.0

losses = []
for i in range(Config.iterations):
    # calc pairwise dists (ND - K1D -> KN shapes for ref)
    d_kn = np.linalg.norm(X_stan - Params.mu[:,None,:], axis=-1)
    # assign cluster based on proximity
    z_n  = d_kn.argmin(axis=0) # N
    losses.append(d_kn.min(axis=0).sum())

    # update cluster means
    z_kn = np.eye(3)[:, z_n] # 3 N
    z_kn/= z_kn.sum(axis=1, keepdims=True) # normalize
    Params.mu = z_kn @ X_stan

fig, ax = plt.subplots(figsize=(10, 8))
ax.plot(range(Config.iterations), losses)
ax.set_ylabel('loss', fontsize=16)
ax.set_xlabel('iteration', fontsize=16)
ax.set_title('Kmeans minimizing total within cluster distance', fontsize=20)
```

```python
plt.yscale('log')
plt.show()

# display the k means
Params.mu = Params.mu * sigma
images = (Params.mu * 255).reshape((Config.K, 28, 28)).astype(np.uint8)
fig, axs = plt.subplots(1, 3, figsize=(18, 6))
for k in range(Config.K):
    axs[k].imshow(images[k], cmap='CMRmap')
    axs[k].set_xlabel(f'Cluster{k}', fontsize=16)
plt.show()
```

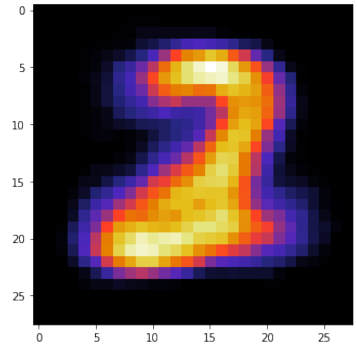




Cluster0



Cluster1



Cluster2