Matheus C. Fernandes
fernandes@g.harvard.edu
CS181-S17

Assignment #4
Due: 5:00pm March 31, 2017

Collaborators:
Bolei Deng
Callin Switzer
Joe Sedlak

Homework 4: Clustering and EM

This homework assignment focuses on different unsupervised learning methods from a theoretical and practical standpoint. In Problem 1, you will explore Hierarchical Clustering and experiment with how the choice of distance metrics can alter the behavior of the algorithm. In Problem 2, you will derive from scratch the full expectation-maximization algorithm for fitting a simple topic model. In Problem 3, you will implement K-Means clustering on a dataset of handwritten images and analyze the latent structure learned by this algorithm.

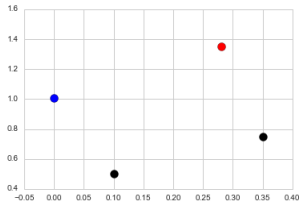
There is a mathematical component and a programming component to this homework. Please submit your PDF and Python files to Canvas, and push all of your work to your GitHub repository. If a question requires you to make any plots, please include those in the writeup.

Hierarchical Clustering [7 pts]

At each step of hierarchical clustering, the two most similar clusters are merged together. This step is repeated until there is one single group. We saw in class that hierarchical clustering will return a different result based on the pointwise-distance and cluster-distance that is is used. In this problem you will examine different choices of pointwise distance (specified through choice of norm) and cluster distance, and explore how these choices change how the HAC algorithm runs on a toy data set.

Problem 1

Consider the following four data points in $\mathbb{R}^2$, belonging to three clusters: the black cluster consisting of $\mathbf{x}_1 = (0.1, 0.5)$ and $\mathbf{x}_2 = (0.35, 0.75)$), the red cluster consisting of $\mathbf{x}_3 = (0.28, 1.35)$, and the blue cluster consisting of $\mathbf{x}_4 = (0, 1.01)$.



Different pointwise distances $d(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_p$ can be used. Recall the definition of the ℓ_1, ℓ_2, and ℓ_∞ norm:

$$\|\mathbf{x}\|_1 = \sum_{j=1}^{m} |x_i| \qquad \|\mathbf{x}\|_2 = \sqrt{\sum_{j=1}^{m} x_i^2} \qquad \|\mathbf{x}\|_\infty = \max_{j \in \{1,\ldots,m\}} |x_j|$$

Also recall the definition of min-distance, max-distance, centroid-distance, and average-distance between two clusters (where $\boldsymbol{\mu}_G$ is the center of a cluster G):

$$\begin{aligned} d_{\text{min}}(G, G') &= \min_{\mathbf{x} \in G, \mathbf{x}' \in G'} d(\mathbf{x}, \mathbf{x}') \\ d_{\text{max}}(G, G') &= \max_{\mathbf{x} \in G, \mathbf{x}' \in G'} d(\mathbf{x}, \mathbf{x}') \\ d_{\text{centroid}}(G, G') &= d(\boldsymbol{\mu}_G, \boldsymbol{\mu}_{G'}) \\ d_{\text{avg}}(G, G') &= \frac{1}{|G||G'|} \sum_{\mathbf{x} \in G} \sum_{\mathbf{x}' \in G'} d(\mathbf{x}, \mathbf{x}') \end{aligned}$$

1. Draw the 2D unit sphere for each norm, defined as $\mathcal{S} = \{\mathbf{x} \in \mathbb{R}^2 : \|\mathbf{x}\| = 1\}$. Feel free to do it by hand, take a picture and include it in your pdf.
2. For each norm ($\ell_1, \ell_2, \ell_\infty$) and each clustering distance, specify which two clusters would be the first to merge.
3. Draw the complete dendrograms showing the order of agglomerations for the ℓ_2 norm and each of the clustering distances.

Solution

Problem 1

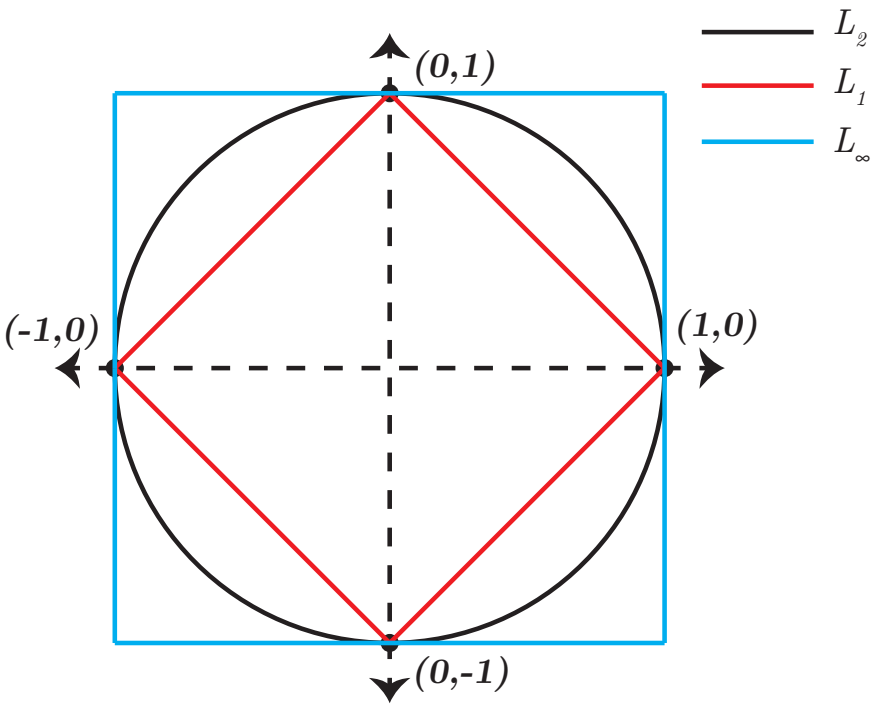


Figure 1

Problem 2

For this question we will subdivide it into the different distance categories, and will analyze the different norms for each of the distance definitions.

(1)

$$d_{\min}(G, G') = \min_{\mathbf{x} \in G, \mathbf{x}' \in G'} d(\mathbf{x}, \mathbf{x}')$$

Using the ℓ_1 norm $d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_1$:
Picking the visually shortest points between the clusters we can boil down to options between black-red or black-blue, and particularly between points x2-x3 or x1-x4, respectively. To do this analysis we compared their norm distances to be:

X_a	X_b	$Dist$
1.	2.	0.5
1.	3.	1.03
1.	4.	0.61
2.	3.	0.67
2.	4.	0.61
3.	4.	0.62

$\boxed{Black - blue}$

Using the ℓ_2 norm $d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_2$:

X_a	X_b	$Dist$
1.	2.	0.35355339
1.	3.	0.86884981
1.	4.	0.51971146
2.	3.	0.60406953
2.	4.	0.43600459
3.	4.	0.44045431

$\boxed{Black - blue}$

Using the Chebyshev norm (ℓ_∞) $d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_\infty$:

X_a	X_b	$Dist$
1.	2.	0.25
1.	3.	0.85
1.	4.	0.51
2.	3.	0.6
2.	4.	0.35
3.	4.	0.34

$\boxed{Red - blue}$

(2)

$$d_{\max}(G, G') = \max_{\mathbf{x} \in G, \mathbf{x}' \in G'} d(\mathbf{x}, \mathbf{x}')$$

Using the ℓ_1 norm $d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_1$:

X_a	X_b	$Dist$
1.	2.	0.5
1.	3.	1.03
1.	4.	0.61
2.	3.	0.67
2.	4.	0.61
3.	4.	0.62

$\boxed{Black - blue}$

Using the ℓ_2 norm $d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_2$:

X_a	X_b	$Dist$
1.	2.	0.35355339
1.	3.	0.86884981
1.	4.	0.51971146
2.	3.	0.60406953
2.	4.	0.43600459
3.	4.	0.44045431

$\boxed{Red - blue}$

Using the Chebyshev norm (ℓ_∞) $d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_\infty$:

X_a	X_b	$Dist$
1.	2.	0.25
1.	3.	0.85
1.	4.	0.51
2.	3.	0.6
2.	4.	0.35
3.	4.	0.34

$\boxed{Red - blue}$

(3)

$$d_{\text{centroid}}(G, G') = d(\boldsymbol{\mu}_G, \boldsymbol{\mu}_{G'})$$

Using the ℓ_1 norm $d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_1$:

X_a	X_b	$Dist$
1.	2.	0.78
1.	3.	0.61
2.	3.	0.62

$\boxed{Black - blue}$

Using the ℓ_2 norm $d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_2$:

$$\begin{bmatrix} X_a & X_b & Dist \\ 1. & 2. & 0.72708321 \\ 1. & 3. & 0.445926 \\ 2. & 3. & 0.44045431 \end{bmatrix}$$

$\boxed{Red - blue}$

Using the Chebyshev norm (ℓ_∞) $d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_\infty$:

$$\begin{bmatrix} X_a & X_b & Dist \\ 1. & 2. & 0.725 \\ 1. & 3. & 0.385 \\ 2. & 3. & 0.34 \end{bmatrix}$$

$\boxed{Red - blue}$

(4)

$$d_{\text{avg}}(G, G') = \frac{1}{|G||G'|} \sum_{\mathbf{x} \in G} \sum_{\mathbf{x}' \in G'} d(\mathbf{x}, \mathbf{x}')$$

Using the ℓ_1 norm $d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_1$:

$$\begin{bmatrix} X_a & X_b & Dist \\ 1. & 2. & 0.85 \\ 1. & 3. & 0.61 \\ 2. & 3. & 0.62 \end{bmatrix}$$

$\boxed{Black - blue}$

Using the ℓ_2 norm $d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_2$:

$$\begin{bmatrix} X_a & X_b & Dist \\ 1. & 2. & 0.73645967 \\ 1. & 3. & 0.47785802 \\ 2. & 3. & 0.44045431 \end{bmatrix}$$

$\boxed{Red - blue}$

Using the Chebyshev norm (ℓ_∞) $d(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'||_\infty$:

$$\begin{bmatrix} X_a & X_b & Dist \\ 1. & 2. & 0.725 \\ 1. & 3. & 0.43 \\ 2. & 3. & 0.34 \end{bmatrix}$$

$\boxed{Red - blue}$

In summary:

	ℓ_1	ℓ_2	ℓ_∞
d_{min}	Black-blue	Black-blue	Red-blue
d_{max}	Black-blue	Red-blue	Red-blue
$d_{centroid}$	Black-blue	Red-blue	Red-blue
d_{avg}	Black-blue	Red-blue	Red-Blue

Problem 3

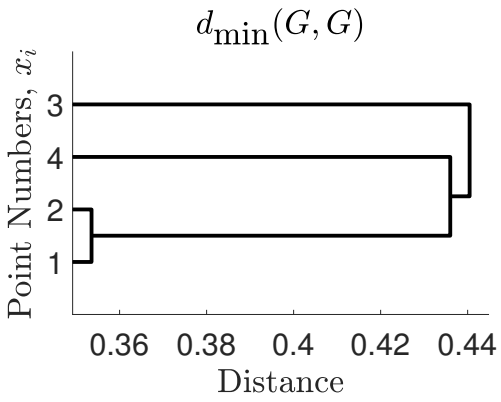


Figure 2

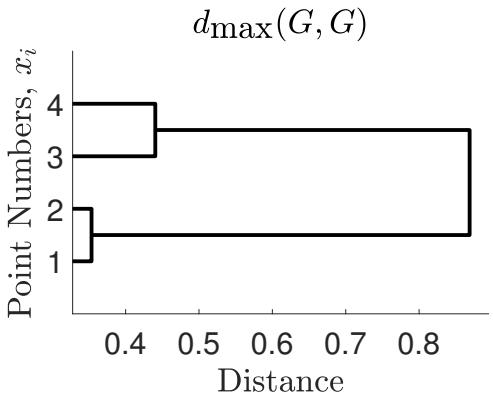


Figure 3

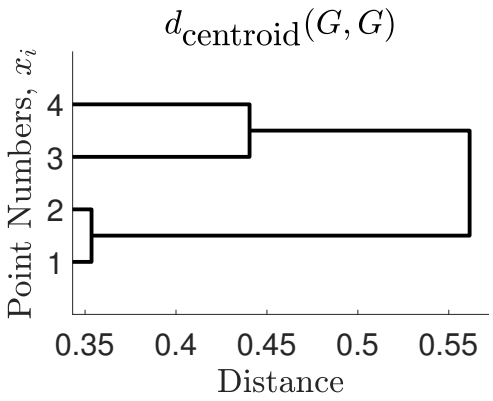


Figure 4

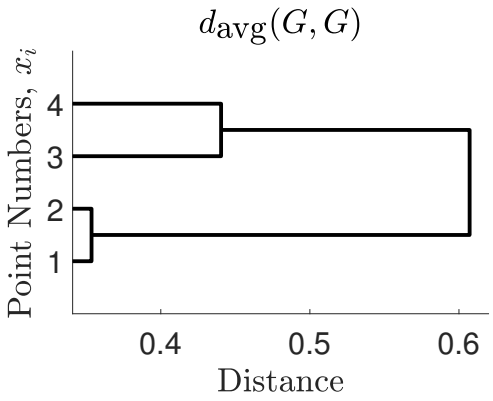


Figure 5

Topic Modeling [15 pts]

In this problem we will explore a simplified version of topic modeling in which each document has just a *single* topic. For this problem, we will assume there are c topics. Each topic $k \in \{1, \ldots, c\}$ will be associated with a vector $\boldsymbol{\beta}_k \in [0,1]^{|\mathcal{V}|}$ describing a distribution over the vocabulary $\mathcal{V}$ with $\sum_{j=1}^{|\mathcal{V}|} \beta_{k,j} = 1$.

Each document $\mathbf{x}_i$ will represented as a bag-of-words $\mathbf{x}_i \in (\mathbb{Z}^{\geq 0})^{|\mathcal{V}|}$, where $x_{i,j}$ is a non-negative integer representing the number of times word j appeared in document i. Document i has n_i word tokens in total. Finally let the (unknown) overall mixing proportion of topics be $\boldsymbol{\theta} \in [0,1]^c$, where $\sum_{k=1}^{c} \theta_k = 1$.

Our generative model is that each of the n documents has a single topic. We encode this topic as a one-hot vector $\mathbf{z}_i \in \{0,1\}^c$ over topics. This one-hot vector is drawn from $\boldsymbol{\theta}$; then, each of the words is drawn from $\boldsymbol{\beta}_{z_i}$. Formally documents are generated in two steps:

$$\begin{aligned} \mathbf{z}_i &\sim \text{Categorical}(\boldsymbol{\theta}) \\ \mathbf{x}_i &\sim \text{Multinomial}(\boldsymbol{\beta}_{\mathbf{z}_i}) \end{aligned}$$

Problem 2

1. Draw the graphical model representation of this problem. Be sure to use the plate notation to indicate repeated random variables and gray nodes to indicate observed variables.
2. **Complete-Data Log Likelihood** Define the complete data for this problem to be $D = \{(\mathbf{x}_i, \mathbf{z}_i)\}_{i=1}^n$.
   - Write out the complete-data (negative) log likelihood.

     $$\mathcal{L}(\boldsymbol{\theta}, \{\boldsymbol{\beta}_k\}_{k=1}^c) = -\ln p(D \mid \boldsymbol{\theta}, \{\boldsymbol{\beta}_k\}_{k=1}^c).$$

   - Explain in one sentence why we cannot directly optimize this loss function.
3. **Expectation Step** Our next step is to introduce a mathematical expression for $\mathbf{q}_i$, the posterior over the hidden topic variables $\mathbf{z}_i$ conditioned on the observed data $\mathbf{x}_i$ with fixed parameters, i.e $p(\mathbf{z}_i | \mathbf{x}_i; \boldsymbol{\theta}, \{\boldsymbol{\beta}_k\}_{k=1}^c)$.
   - Write down and simplify the expression for $\mathbf{q}_i$.
   - Give an algorithm for calculating $\mathbf{q}_i$ for all i, given the observed data $\{\mathbf{x}_i\}_{i=1}^n$ and settings of the parameters $\boldsymbol{\theta}$ and $\{\boldsymbol{\beta}_k\}_{k=1}^c$.
4. **Maximization Step** Using the $\mathbf{q}_i$ estimates from the Expectation Step, derive an update for maximizing the expected complete data log likelihood in terms of $\boldsymbol{\theta}$ and $\{\boldsymbol{\beta}_k\}_{k=1}^c$.
   - Derive an expression for the expected complete-data log likelihood in terms of $\mathbf{q}_i$.
   - Find an expression for $\boldsymbol{\theta}$ that maximizes this expected complete-data log likelihood. You may find it helpful to use Lagrange multipliers in order to force the constraint $\sum \theta_k = 1$. Why does this optimized $\boldsymbol{\theta}$ make intuitive sense?
   - Apply a similar argument to find the value of the $\boldsymbol{\beta}_k$'s that maximizes the expected complete-data log likelihood.

Solution

Problem 1

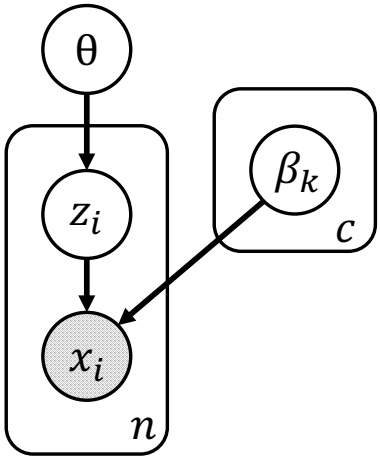


Figure 6: Graphical Representation of Problem 2

Problem 2

The complete-data (negative) log likelihood is given as:

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}, \{\boldsymbol{\beta}_k\}_{k=1}^c) &= -\ln p(D \mid \boldsymbol{\theta}, \{\boldsymbol{\beta}_k\}_{k=1}^c) \\
&= -\ln \prod_{i=1}^{n} p(\{\mathbf{x}_i, \mathbf{z}_i\} | \theta, \{\boldsymbol{\beta}_k\}_{k=1}^c) \\
&= -\ln \prod_{i=1}^{n} p(\mathbf{x}_i | \mathbf{z}_i; \theta, \{\boldsymbol{\beta}_k\}_{k=1}^c)) p(\mathbf{z}_i; \theta, \{\boldsymbol{\beta}_k\}_{k=1}^c)) \\
&= -\ln \prod_{i=1}^{n} \left(\prod_{j=1}^{|\mathcal{V}|} \prod_{k=1}^{c} \beta_{k,j}^{x_{ij} z_{ik}} \times \prod_{k=1}^{c} \theta_k^{z_{ik}} \right) \\
&= -\sum_{i=1}^{n} \left(\sum_{j=1}^{|\mathcal{V}|} \sum_{k=1}^{c} \ln \beta_{k,j}^{x_{ij} z_{ik}} + \sum_{k=1}^{c} \ln \theta_k^{z_{ik}} \right) \\
&= -\sum_{i=1}^{n} \left(\sum_{j=1}^{|\mathcal{V}|} \sum_{k=1}^{c} x_{ij} z_{ik} \ln \beta_{k,j} + \sum_{k=1}^{c} z_{ik} \ln \theta_k \right) \\
&\boxed{= -\sum_{i=1}^{n} \sum_{k=1}^{c} z_{ik} \left(\sum_{j=1}^{|\mathcal{V}|} x_{ij} \ln \beta_{k,j} + \ln \theta_k \right)}
\end{aligned}
$$

The reason why we cannot directly optimize this loss function is we don't know the latent variable z_i.

Problem 3

For this problem we want to know the posterior $\mathbf{q}_i$ which is given as

$$\mathbf{q}_i = p(\mathbf{z}_i|\mathbf{x}_i; \boldsymbol{\theta}, \{\boldsymbol{\beta}_k\}_{k=1}^c).$$

However, it may be more useful to work with each individual element of $\mathbf{q}_i$ as q_{ik}. To do this, we can take the solution from above and normalize it by the sum of all of the values such that:

$$q_{ik} = \frac{\theta_k \prod_{j=1}^{|\mathcal{V}|} \beta_{k,j}^{x_{ij}}}{\sum_{k=1}^c \theta_k \prod_{j=1}^{|\mathcal{V}|} \beta_{k,j}^{x_{ij}}} \quad \text{where } \mathbf{q}_i = q_{ik} \forall k \in [1, c]$$

Because each value q_{ik} depends on every other value of q, that means we cannot compute this naively as is. To avoid this issue, I would recommend an algorithm that solves for all values of q_{ik} without the denominator, then computes the actual sum and divides all of the values by that sum. In other words:

(1) Compute the numerator for all q_{ik}

(2) Compute the sum of the denominator with the values from the previous step

(3) Divide the numerator for all q_{ik} by the denominator sum computed in the previous step.

Problem 4

We know that for this problem we cannot solve directly for $\mathbf{z}_i$ in the complete-data log likelihood, thus we must use our expectation step to infer the $\mathbf{z}_i$ as $\mathbf{q}_i$.

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}, \{\boldsymbol{\beta}_k\}_{k=1}^c) &= -\sum_{i=1}^n \sum_{k=1}^c z_{ik} \left(\sum_{j=1}^{|\mathcal{V}|} x_{ij} \ln \beta_{k,j} + \ln \theta_k \right) \\
&= -\sum_{i=1}^n \sum_{k=1}^c q_{ik} \left(\sum_{j=1}^{|\mathcal{V}|} x_{ij} \ln \beta_{k,j} + \ln \theta_k \right) \\
&= -\sum_{i=1}^n \sum_{k=1}^c \left(\sum_{j=1}^{|\mathcal{V}|} x_{ij} \ln \beta_{k,j} + \ln \theta_k \right) \frac{\theta_k \prod_{j=1}^{|\mathcal{V}|} \beta_{k,j}^{x_{ij}}}{\sum_{k=1}^c \theta_k \prod_{j=1}^{|\mathcal{V}|} \beta_{k,j}^{x_{ij}}}
\end{aligned}$$

To find an expression for θ_k that maximizes the expected complete-data log likelihood we introduce the Lagrangian function to optimize for the expected complete-data log likelihood while obeying the constraint on θ_k. To do this, we have the following Lagrangian:

$$\mathscr{L}(\theta_k, \lambda) = -\sum_{i=1}^n \sum_{k=1}^c q_{ik} \left(\sum_{j=1}^{|\mathcal{V}|} x_{ij} \ln \beta_{k,j} + \ln \theta_k \right) + \lambda \left(\sum_{k=1}^c \theta_k - 1 \right)$$

Taking the derivative of the lagrangian with respect to the λ parameter and setting it to zero gives us:

$$\frac{\partial}{\partial \lambda} \mathscr{L}(\theta_k, \lambda) = \sum_{k=1}^c \theta_k - 1 = 0$$

$$\frac{\partial}{\partial \theta_k}\mathscr{L}(\theta_k, \lambda) = -\sum_{i=1}^{n}\frac{1}{\theta_k}q_{ik} + \lambda = 0 \Rightarrow \theta_k = \frac{1}{\lambda}\sum_{i=1}^{n} q_{ik}$$

Plugging the above into our constraint $\sum_{k=1}^{c}\theta_k - 1 = 0$, we can solve for λ:

$$\lambda = \sum_{k=1}^{c}\sum_{i=1}^{n} q_{ik} = n$$

So, plugging the above in to our original equation for θ_k, our final expression thus becomes:

$$\boxed{\theta_k = \frac{1}{n}\sum_{i=1}^{n} q_{ik}}$$

Now to find β, we take a very similar approach and begin by introducing the Lagrangian

$$\mathscr{L}(\beta_k, \lambda_k') = -\sum_{i=1}^{n}\sum_{k=1}^{c} q_{ik}\left(\sum_{j=1}^{|\mathcal{V}|} x_{ij}\ln\beta_{k,j} + \ln\theta_k\right) + \sum_{k=1}^{c}\lambda_k'\left(\sum_{j=1}^{|\mathcal{V}|}\beta_{k,j} - 1\right)$$

Now, taking the respective derivatives, we get:

$$\frac{\partial}{\partial \lambda_k'}\mathscr{L}(\beta_{k,j}, \lambda_k') = \left(\sum_{j=1}^{|\mathcal{V}|}\beta_{k,j} - 1\right) = 0$$

$$\frac{\partial}{\partial \beta_{k,j}}\mathscr{L}(\beta_{k,j}, \lambda_k') = -\sum_{i=1}^{n} q_{ik}x_{ij}\frac{1}{\beta_{k,j}} + \lambda_k' = 0 \Rightarrow \frac{1}{\lambda_k'}\sum_{i=1}^{n} q_{ik}x_{ij} = \beta_{k,j}$$

Plugging the above into our constraint $\left(\sum_{j=1}^{|\mathcal{V}|}\beta_{k,j} - 1\right) = 0$, we can solve for λ_k':

$$\left(\frac{1}{\lambda_k'}\sum_{j=1}^{|\mathcal{V}|}\sum_{i=1}^{n} q_{ik}x_{ij} - 1\right) = 0 \Rightarrow \lambda_k' = \sum_{j=1}^{|\mathcal{V}|}\sum_{i=1}^{n} q_{ik}x_{ij}$$

$$\boxed{\beta_{k,j} = \frac{\sum_{i=1}^{n} q_{ik}x_{ij}}{\sum_{j=1}^{|\mathcal{V}|}\sum_{i=1}^{n} q_{ik}x_{ij}}}$$

K-Means [15 pts]

For this problem you will implement K-Means clustering from scratch. Using `numpy` is fine, but don't use a third-party machine learning implementation like `scikit-learn`. You will then apply this approach to clustering of image data.

We have provided you with the MNIST dataset, a collection of handwritten digits used as a benchmark of image recogntion (you can learn more about the data set at http://yann.lecun.com/exdb/mnist/). The MNIST task is widely used in supervised learning, and modern algorithms with neural networks do very well on this task.

Here we will use MNIST unsupervised learning. You have been given representations of 6000 MNIST images, each of which are 28×28 greyscale handwritten digits. Your job is to implement K-means clustering on MNIST, and to test whether this relatively simple algorithm can cluster similar-looking images together.

Problem 3

The given code loads the images into your environment as a 6000x28x28 array.

- Implement K-means clustering from different random initializations and for several values of K using the ℓ_2 norm as your distance metric. (You should feel free to explore other metrics than the ℓ_2 norm, but this is strictly optional.) Compare the K-means objective for different values of K and across random initializations.
- For three different values of K, and a couple of random restarts for each, show the mean images for each cluster (i.e., for the cluster prototypes), as well as the images for a few representative images for each cluster. You should explain how you selected these representative images. To render an image, use the pyplot `imshow` function.
- Are the results wildly different for different restarts and/or different values of K? For one of your runs, plot the K-means objective function as a function of iteration and verify that it never increases.

As in past problem sets, please include your plots in this document. (There may be several plots for this problem, so feel free to take up multiple pages.)

Solution

- *Implement K-means clustering from different random initializations and for several values of K using the ℓ_2 norm as your distance metric. (You should feel free to explore other metrics than the ℓ_2 norm, but this is strictly optional.) Compare the K-means objective for different values of K and across random initializations.*

  asdfasdf

- *For three different values of K, and a couple of random restarts for each, show the mean images for each cluster (i.e., for the cluster prototypes), as well as the images for a few representative images for each cluster. You should explain how you selected these representative images. To render an image, use the pyplot `imshow` function.*
- *Are the results wildly different for different restarts and/or different values of K? For one of your runs, plot the K-means objective function as a function of iteration and verify that it never increases.*

Problem 4 (Calibration, 1pt)
Approximately how long did this homework take you to complete?

~ 13 hours

Problem 3 Code – K-Means Clustering

```
# CS 181, Spring 2017
# Homework 4: Clustering
# Name: Matheus C Fernandes
# Email: fernandes@seas.harvard.edu

import numpy as np
import matplotlib.pyplot as plt
import random
import matplotlib.image as mpimg


class KMeans(object):
    # K is the K in KMeans
    def __init__(self, K, D):
        self.K = K
        self.D = D

    def __norm(self, X):
        return np.linalg.norm(X)  # np.sum(np.sum(X**2.))**(1./2.)

    def varyingK(self, X, numberOfRestarts, ks):
        for k in ks:
            self.K = k
            self.random_Restarts(X, numberOfRestarts)

    def random_Restarts(self, X, numberOfRestarts):
        Loss = [];
        ct = []
        for r in xrange(numberOfRestarts):
            self.randomR = r
            Loss.append(self.fit(X))
            ct.append(r)
        plt.figure()
        plt.plot(ct, Loss, lw=3)
        plt.title(r'Loss vs. RandomRestart K={}'.format(int(self.K)))
        plt.xlabel(r'Random Restart Number')
        plt.ylabel(r'Final Loss')
        plt.savefig("plots/RR_K{}.pdf".format(int(self.K)), dpi=100)

        self.plot_representative_images(self.D, X)
        self.plot_Loss()

    # X is a (N x 28 x 28) array where 28x28 is the dimensions of each of the N images.
    def fit(self, X):
        K = self.K

        mu = np.zeros((K, X.shape[1], X.shape[2]))
        lenXi = X.shape[0]

        # initialize the variable mu within random points of X
        for k in xrange(int(K)):
            beginInt = random.randint(0, X.shape[0] - 1)
            mu[k, :, :] = X[beginInt, :, :]

        # usefull definitions for convergence
        converged = True
        ct = 0
        Loss = []
        ctAll = []

        print 'Fitting Began K = {}, Random Iteration = {}'.format(self.K, self.randomR + 1)

```

```
        while converged:
            ct += 1
            r = np.zeros((X.shape[0], K))  # resetting the R matrix to have all 0's

            # obtaining the R-matrix based on the differences between mu and the X's
            for i in xrange(lenXi):
                allMu = np.array([self.__norm(X[i, :, :] - mu[k, :, :]) for k in range(int(K))])
                r[i, np.argmin(allMu)] = 1.

            LossSum = 0
            # updating the mean matrices
            for k in xrange(K):
                nk = sum(r[:, k])
                if nk != 0:
                    mu[k] = np.sum([(1. / nk) * r[i, k] * X[i, :, :] for i in range(lenXi)], axis
=0)
                # calculating the loss based on the equation in the slides
                LossSum += np.sum([r[i, k] * self.__norm(X[i, :, :] - mu[k, :, :]) for i in range
(lenXi)])

            # updating master list of all loss and counts
            Loss.append(LossSum)
            ctAll.append(ct)

            # printing status and checking convergence for breaking while loop
            if ct != 1 and abs(Loss[-1] - Loss[-2]) < 1: converged = False; print '\nConverged!\n
'

        # exporting data/information to self
        self.Loss = np.array(Loss)
        self.ctAll = np.array(ctAll)
        self.mu = mu

        return min(Loss)

    def plot_Loss(self):
        plt.figure()
        plt.plot(self.ctAll, self.Loss, lw=3)
        plt.title(r'Loss vs. Iteration K={}'.format(self.K))
        plt.xlabel(r'Iteration')
        plt.ylabel(r'Loss')
        plt.savefig("plots/L-K{}.pdf".format(int(self.K)), dpi=100)

    def plot_representative_images(self, D, X):
        import matplotlib.gridspec as gridspec

        muImages = self.mu
        RepImages = self.get_representative_images(D, X)

        fig = plt.figure()
        gs = gridspec.GridSpec(D + 1, muImages.shape[0])
        ax = [plt.subplot(gs[i]) for i in xrange((D + 1) * (muImages.shape[0]))]

        gs.update(hspace=0)

        ctr = 0
        for muCount in xrange(muImages.shape[0]):
            ax[ctr].imshow(muImages[muCount].reshape(28, 28), cmap='Greys_r')
            ax[ctr].axis('off')
            ctr += 1
        for DCount in xrange(D):
            for muCount in xrange(muImages.shape[0]):
                ax[ctr].imshow(RepImages[muCount, DCount, :, :].reshape(28, 28), cmap='Greys_r')
                ax[ctr].axis('off')
                ctr += 1
```

```
125             plt.suptitle(r'K={} and D={}'.format(int(muImages.shape[0]), int(D)), size=20)
126             plt.savefig("plots/KD-K{}.pdf".format(int(self.K)), dpi=100)
127 
128         # This should return the arrays for K images. Each image should represent the mean of each of
            the fitted clusters.
129         def get_mean_images(self):
130             return self.mu
131 
132         # This should return the arrays for D images from each cluster that are representative of the
            clusters.
133         def get_representative_images(self, D, X):
134             mu = self.mu
135             ImageDiff = np.zeros((mu.shape[0], X.shape[0]))
136             ImageClosest = np.zeros((mu.shape[0], D, X.shape[1], X.shape[2]))
137 
138             for k in xrange(mu.shape[0]):
139                 ImageDiff[k] = [self.__norm(X[i, :, :] - mu[k, :, :]) for i in range(X.shape[0])]
140                 SortIndex = np.array([i[0] for i in sorted(enumerate(ImageDiff[k]), key=lambda x: x
        [1])])
141                 ImageClosest[k] = X[SortIndex[:D], :, :]
142 
143             return ImageClosest
144 
145         # img_array should be a 2D (square) numpy array.
146         # Note, you are welcome to change this function (including its arguments and return values)
            to suit your needs.
147         # However, we do ask that any images in your writeup be grayscale images, just as in this
            example.
148         def create_image_from_array(self, img_array):
149             plt.figure()
150             plt.imshow(img_array, cmap='Greys_r')
151             plt.show()
152             return
153 
154 
155 # This line loads the images for you. Don't change it!
156 pics = np.load("images.npy", allow_pickle=False)
157 
158 # You are welcome to change anything below this line. This is just an example of how your code
        may look.
159 # That being said, keep in mind that you should not change the constructor for the KMeans class,
160 # though you may add more public methods for things like the visualization if you want.
161 # Also, you must cluster all of the images in the provided dataset, so your code should be fast
        enough to do that.
162 
163 K = 10
164 D = 10
165 KMeansClassifier = KMeans(K,D)
166 KMeansClassifier.varyingK(pics,10,[5,10,15])
```

Listing 1: Code for problem 3