Kendrick Lo
klo@g.harvard.edu
CS181-S16

Assignment #4
Due: 5:00pm Feb 1, 2016

Collaborators: A. Lee, TAs (OHs)

Homework 4: Clustering

There is a mathematical component and a programming component to this homework. Please submit ONLY your PDF 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.

Problem 1 (The Curse of Dimensionality, 4pts)
In d dimensions, consider a hypersphere of unit radius, centered at zero, which is inscribed in a hypercube, also centered at zero, with edges of length two. What fraction of the hypercube's volume is contained within the hypersphere? Write this as a function of d. What happens when d becomes large?

Solution

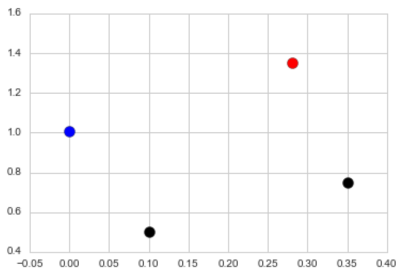
A d-dimensional hypercube with edges of length two has volume 2^d (see https://en.wikipedia.org/wiki/Hypercube). The formula for the volume of a hypersphere with unit radius is: $\frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}$ (see https://en.wikipedia.org/wiki/Volume_of_an_n-ball). Therefore, the fraction of the hypercube's volume contained within the hypersphere as a function of d is:

$$\frac{\frac{\pi^{d/2}}{\Gamma(\frac{d}{2}+1)}}{2^d} = \frac{\frac{(\pi^{1/2})^d}{\Gamma(\frac{d}{2}+1)}}{2^d} = \frac{(\frac{\pi^{1/2}}{2})^d}{\Gamma(\frac{d}{2}+1)}$$

The numerator is $O(2^d)$ while the denominator is $O(d!)$. As d becomes very large, the denominator grows faster than the numerator, and therefore the above ratio (and thus the fraction of the hypercube's volume contained within the hypersphere) approaches 0. Intuitively, the hypersphere reduces to a point within the hypercube volume at high dimensions.

Problem 2 (Norms, Distances, and Hierarchical Clustering, 5 pts)

Consider the following four data points, belonging to three clusters: the black cluster $((x_1, y_1) = (0.1, 0.5)$ and $(x_2, y_2) = (0.35, 0.75))$, the red cluster $(x_3, y_3) = (0.28, 1.35)$ cluster, and the blue cluster $(x_4, y_4) = (0, 1.01)$.



At each step of hierarchical clustering, the two most similar (or least dissimilar) clusters are merged together. This step is repeated until there is one single group. Different distances can be used to measure group dissimilarity. Recall the definition of the l_1, l_2, and l_∞ norm:

- For $\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$
- For $\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^n x_i^2}$
- For $\mathbf{x} \in \mathbb{R}^n$, $\|\mathbf{x}\|_\infty = \max_{i=1}^n |x_i|$

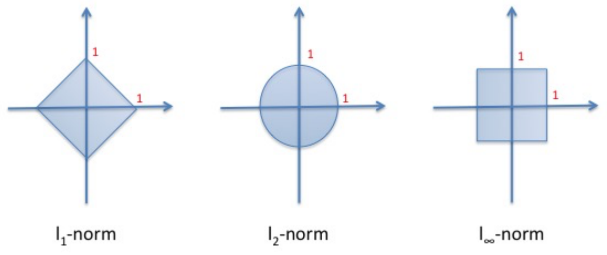
Also recall the definition of single-link distance, complete-link distance, and average-link distance between two clusters:

- Single-link clustering: for clusters G and H, $d_S(G, H) = \min_{i \in G, j \in H} d(i, j)$
- Complete-link clustering: for clusters G and H, $d_C(G, H) = \max_{i \in G, j \in H} d(i, j)$
- Average-link clustering: for clusters G and H, $d_A(G, H) = \frac{1}{|G||H|} \sum_{i \in G} \sum_{j \in H} d(i, j)$

Warm up question. Draw the 2D unit sphere for each norm, defined as $\mathcal{S} = \{x \in \mathbb{R}^2 : \|x\| = 1\}$. Feel free to do it by hand, take a picture and include it in your pdf.

Main question. For each norm (l_1, l_2, l_∞) and each clustering method (single, complete, or average link clustering), specify which 2 clusters would be the first to merge.

Solution



l_1 norm: the sphere is a diamond with sides of equal length, with the distance from the origin to the vertex being 1; l_2 norm: unit circle (r=1); l_∞ norm: square with its edges crossing the axes at (0, 1), (1, 0), (-1, 0) and (0, -1) as shown.

We now work through each clustering method sequentially to determine which 2 clusters would be the first to merge.

1. Single-link clustering (look for closest neighbors in other clusters)
   (a) l_1: **BLUE and BLACK**
   (*Red-Blue: $|1.35 - 1.01| + |0.28 - 0| = 0.62$,
   Red-Black1: $|1.35 - 0.5| + |0.28 - 0.1| = 1.03$,
   *Red-Black2: $|1.35 - 0.75| + |0.28 - 0.35| = 0.67$,
   *Blue-Black1: $|1.01 - 0.5| + |0 - 0.1| = 0.61$,
   *Blue-Black2: $|1.01 - 0.75| + |0 - 0.35| = 0.61$)
   (b) l_2: **BLUE and BLACK**
   (*Red-Blue: $\sqrt{(1.35 - 1.01)^2 + (0.28 - 0)^2} = 0.440$,
   Red-Black1: $\sqrt{(1.35 - 0.5)^2 + (0.28 - 0.1)^2} = 0.869$,
   *Red-Black2: $\sqrt{(1.35 - 0.75)^2 + (0.28 - 0.35)^2} = 0.604$,
   Blue-Black1: $\sqrt{(1.01 - 0.5)^2 + (0 - 0.1)^2} = 0.520$,
   *Blue-Black2: $\sqrt{(1.01 - 0.75)^2 + (0 - 0.35)^2} = 0.436$)
   (c) l_∞: **RED and BLUE**
   (*Red-Blue: $max(|1.35 - 1.01|, |0.28 - 0|) = 0.34$,
   Red-Black1: $max(|1.35 - 0.5|, |0.28 - 0.1|) = 0.85$,
   *Red-Black2: $max(|1.35 - 0.75|, |0.28 - 0.35|) = 0.60$,
   Blue-Black1: $max(|1.01 - 0.5|, |0 - 0.1|) = 0.51$,
   *Blue-Black2: $max(|1.01 - 0.75|, |0 - 0.35|) = 0.35$)
2. Complete-link clustering (look for furthest neighbors in other clusters)
   (a) l_1: **BLUE and BLACK**
   (*Red-Blue: $|1.35 - 1.01| + |0.28 - 0| = 0.62$,
   *Red-Black1: $|1.35 - 0.5| + |0.28 - 0.1| = 1.03$,
   Red-Black2: $|1.35 - 0.75| + |0.28 - 0.35| = 0.67$,
   *Blue-Black1: $|1.01 - 0.5| + |0 - 0.1| = 0.61$,
   *Blue-Black2: $|1.01 - 0.75| + |0 - 0.35| = 0.61$)
   (b) l_2: **RED and BLUE**
   (*Red-Blue: $\sqrt{(1.35 - 1.01)^2 + (0.28 - 0)^2} = 0.440$,
   *Red-Black1: $\sqrt{(1.35 - 0.5)^2 + (0.28 - 0.1)^2} = 0.869$,
   Red-Black2: $\sqrt{(1.35 - 0.75)^2 + (0.28 - 0.35)^2} = 0.604$,
   *Blue-Black1: $\sqrt{(1.01 - 0.5)^2 + (0 - 0.1)^2} = 0.520$,
   Blue-Black2: $\sqrt{(1.01 - 0.75)^2 + (0 - 0.35)^2} = 0.436$)
   (c) l_∞: **RED and BLUE**
   (*Red-Blue: $max(|1.35 - 1.01|, |0.28 - 0|) = 0.34$,
   *Red-Black1: $max(|1.35 - 0.5|, |0.28 - 0.1|) = 0.85$,
   Red-Black2: $max(|1.35 - 0.75|, |0.28 - 0.35|) = 0.60$,
   *Blue-Black1: $max(|1.01 - 0.5|, |0 - 0.1|) = 0.51$,
   Blue-Black2: $max(|1.01 - 0.75|, |0 - 0.35|) = 0.35$)

3. Average-link clustering (get average distance to points in other clusters)

   (a) l_1: **BLUE and BLACK**
   (*Red-Blue: $|1.35 - 1.01| + |0.28 - 0| = 0.62$,
   Red-Black1: $|1.35 - 0.5| + |0.28 - 0.1| = 1.03$,
   Red-Black2: $|1.35 - 0.75| + |0.28 - 0.35| = 0.67$,
   *Red-Black-avg: $(1.03 + 0.67)/2 = 0.85$
   Blue-Black1: $|1.01 - 0.5| + |0 - 0.1| = 0.61$,
   Blue-Black2: $|1.01 - 0.75| + |0 - 0.35| = 0.61$)
   *Blue-Black-avg: 0.61)

   (b) l_2: **RED and BLUE**
   (*Red-Blue: $\sqrt{(1.35 - 1.01)^2 + (0.28 - 0)^2} = 0.440$,
   Red-Black1: $\sqrt{(1.35 - 0.5)^2 + (0.28 - 0.1)^2} = 0.869$,
   Red-Black2: $\sqrt{(1.35 - 0.75)^2 + (0.28 - 0.35)^2} = 0.604$,
   *Red-Black-avg: $(0.869 + 0.604)/2 = 0.737$,
   Blue-Black1: $\sqrt{(1.01 - 0.5)^2 + (0 - 0.1)^2} = 0.520$,
   Blue-Black2: $\sqrt{(1.01 - 0.75)^2 + (0 - 0.35)^2} = 0.436$,
   *Blue-Black-avg: $(0.520 + 0.436)/2 = 0.478$)

   (c) l_∞: **RED and BLUE**
   (*Red-Blue: $max(|1.35 - 1.01|, |0.28 - 0|) = 0.34$,
   Red-Black1: $max(|1.35 - 0.5|, |0.28 - 0.1|) = 0.85$,
   Red-Black2: $max(|1.35 - 0.75|, |0.28 - 0.35|) = 0.60$,
   *Red-Black-avg: $(0.85 + 0.60)/2 = 0.725$,
   Blue-Black1: $max(|1.01 - 0.5|, |0 - 0.1|) = 0.51$,
   Blue-Black2: $max(|1.01 - 0.75|, |0 - 0.35|) = 0.35$)
   *Blue-Black-avg: $(0.51 + 0.35)/2 = 0.43$,

K-Means [15 pts]

Implement K-Means clustering from scratch.[1]. You have been provided with the MNIST dataset. You can learn more about it 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. We can also use MNIST for interesting unsupervised tasks. You are given representations of 6000 MNIST images, each of which are 28×28 handwritten digits. In this problem, you will implement K-means clustering on MNIST, to show how this relatively simple algorithm can cluster similar-looking images together quite well.

Problem 3 (K-means, 15pts)

The given code loads the images into your environment as a 6000x28x28 array. Implement K-means clustering on it for a few different values of K, and show results from the fit. Show the mean images for each class, and by selecting a few representative images for each class. You should explain how you selected these representative images. To render an image, use the numpy imshow function, which the distribution code gives an example of. Use squared norm as your distance metric. You should feel free to explore other metrics along with squared norm if you are interested in seeing the effects of using those. Also, your code should use the entire provided 6000-image dataset (which, by the way, is only 10% of the full MNIST set).

Are the results wildly different for different restarts and/or different K? Plot the K-means objective function as a function of iteration and verify that it never increases.

Finally, implement K-means++ and see if gives you more satisfying initializations (and final results) for K-means. Explain your findings.

As in past problem sets, please include your plots in this document. There may be tons of plots for this problem, so feel free to take up multiple pages, as long as it is organized.

Solution

The K-means algorithm (along with K-means++) was implemented, and the modified problem3.py has been pushed to Github.

We ran the algorithm on a few different values of K, and show results from the fits below.

First, we provide the final "mean" images for each cluster, for each run. Note that the "mean" is simply an average of values for each "pixel", and therefore **it is not necessarily the case (and in fact it is probably highly unlikely) that the mean image corresponds exactly to an actual image in the data set**.

Second, we selected a few representative images for each cluster. We fixed the number of representative images to display (9), and then we showed: (a) the image that is closest to the mean of the cluster to which it belongs, (b) the image that is furthest from the mean of the cluster to which it belongs, (c) and remaining images at intermediate distances from the mean of the respective cluster. **This allows us to see the range of images that have been assigned to the cluster.** We noticed that images that were further from the cluster mean looked quite different from a handful of those closer to the cluster; we might reasonably expect that the decision to include these "furthest" images in the given cluster may have been a borderline one.

We used squared norm as our metric, although we wrote a separate function for the distance calculation that allowed us to experiment with different distance metrics.

We note from the images below that the means for K=10 (we assumed there were ten possible digits, suggesting that this number of clusters would be ideal), were not all that unique to our eye. We were expecting digits to be separated by number, but we can see the clustering algorithm is comparing other properties (e.g. the difference in slant between and thickness of strokes, the number and positions of curved strokes, etc.) In this regard, certain numbers were clearly represented in a cluster for the most part (e.g. 0's with its

[1]That is, don't use a third-party machine learning implementation like `scikit-learn`; `numpy` is fine.

characteristic center hole, thick diagonal ”1”s, 6’s, and 2s), while others showed some mixing (e.g. 4s, 8s, 3s).

For smaller numbers of K (e.g. we tried K=6), we had more digits that took up similar area seemingly grouped together; for larger numbers of K (e.g. we tried K=15), it seemed we were able to capture sets of digits more readily (although occasionally we would have two of the same ordinal number that were drawn differently belonging to two different clusters). This is not necessarily a bad thing; we may want to distinguish between a slanted ”1” and a vertical ”1” for example.

We used the entire 6000-image data set in our simulation.

Moving back to the K=10 simulation, for example, we ran this several times and compared the number of images in each of the final clusters after convergence:

```
distribution (images per cluster)
[465, 475, 513, 513, 574, 602, 608, 650, 754, 846], J = 15131825362.7
[394, 477, 480, 490, 589, 599, 635, 708, 732, 896], J = 15166461159.2
[414, 458, 476, 509, 578, 620, 659, 729, 731, 826], J = 15059564650.9
[409, 463, 482, 518, 594, 605, 649, 717, 724, 839], J = 15060286409.4
[379, 471, 481, 513, 554, 572, 677, 727, 769, 857], J = 15211523245.2
[244, 346, 414, 552, 590, 592, 693, 731, 749, 1089], J = 15144595015.2
[420, 449, 475, 508, 572, 620, 659, 730, 737, 830], J = 15059644935.4
[417, 455, 477, 508, 581, 623, 650, 729, 729, 831], J = 15059552398.9
[415, 458, 476, 509, 578, 621, 656, 729, 733, 825], J = 15059559590.7
[420, 454, 473, 511, 583, 615, 672, 723, 737, 812], J = 15060459206.9
```

As the counts may vary quite widely, this provides evidence that upon convergence **we are NOT guaranteed a unique solution (i.e. the global optimum)**. The solution we get is highly dependent on where we start; accordingly, we found that it may be best to run the simulation over multiple iterations, and then use the absolute value of the objective function (J) to choose a final model, in the event we started with a poor initial set of means just by chance.

With respect to the objective function, we plotted its value versus iteration number and confirmed visually that the objective function never increases.

Finally, we implemented K-means++ to see if we would get any improvement. In K-means++, we sequentially choose images to be the initial cluster means, based on the distance from already-chosen cluster means, in a manner that on average allows new cluster mean locations to be chosen that are far away from already-chosen cluster means.

Unlike the original algorithm, we could actually see the images being chosen as the starting cluster means, rather than some blurry picture, so that was more intuitive. Also, the K-means++ implementation tended to start with a lower initial value for the objective function, and many times (although not always), the objective function drops relatively much more quickly to a lower level in the first few iterations and/or converges more quickly, compared to the original algorithm. That said, the visual results and the final values of the objective function seemed quite comparable; it may be that since the data set is very large, means calculated from a large number of random starting points may be unlikely to be close together by chance, and we ran the original algorithm multiple times over anyway to allow us to select a better model.

Also note that finding **the** grand optimum is still not guaranteed when using the K-means++ algorithm.

Figures for K=10 simulation.

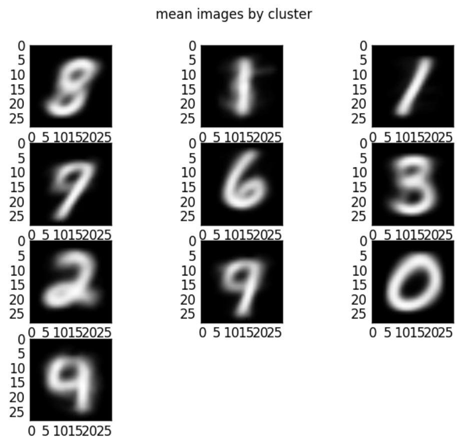


Figure 1: K=10, final mean images

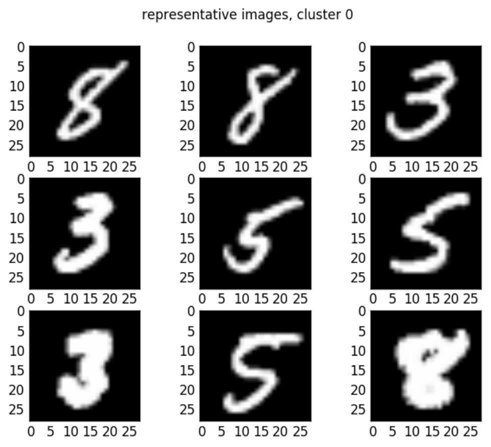


Figure 2: K=10, cluster 0 representative images

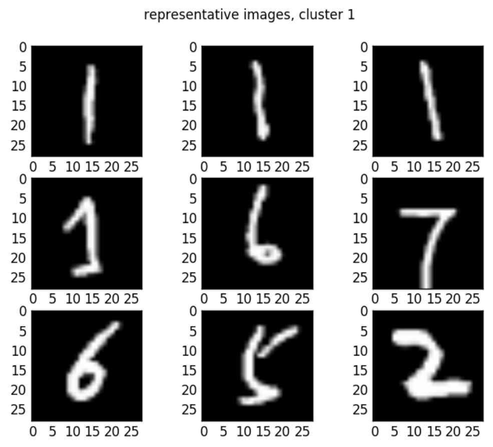


Figure 3: K=10, cluster 1 representative images

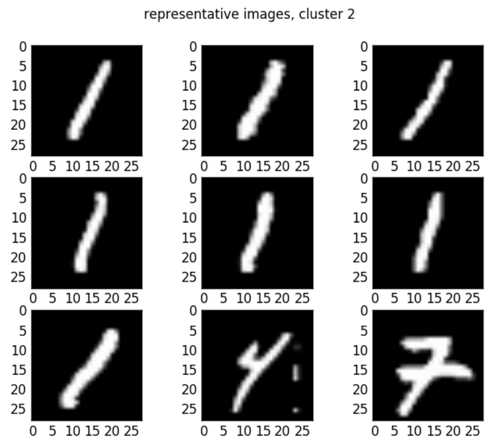


Figure 4: K=10, cluster 2 representative images

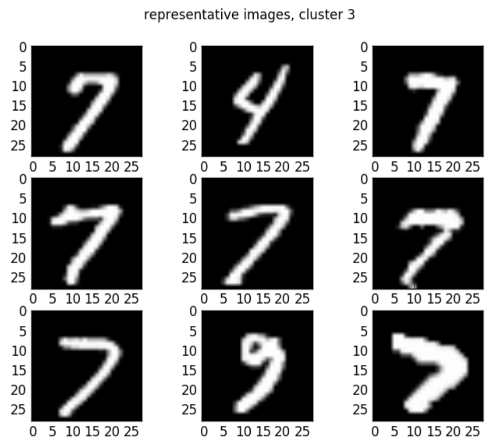


Figure 5: K=10, cluster 3 representative images

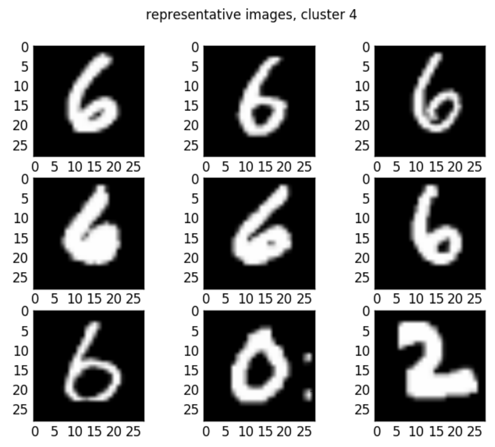


Figure 6: K=10, cluster 4 representative images

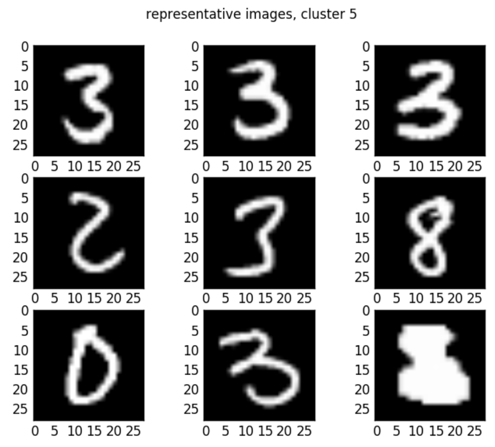


Figure 7: K=10, cluster 5 representative images

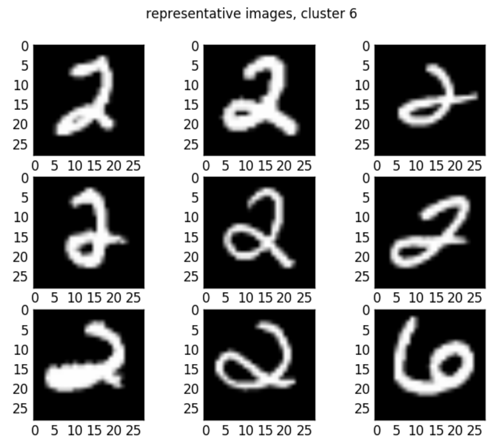


Figure 8: K=10, cluster 6 representative images

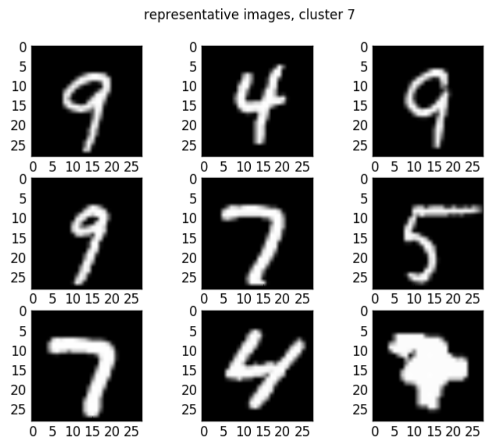


Figure 9: K=10, cluster 7 representative images

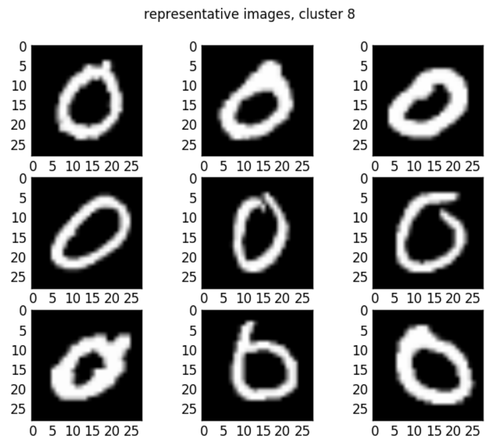


Figure 10: K=10, cluster 8 representative images

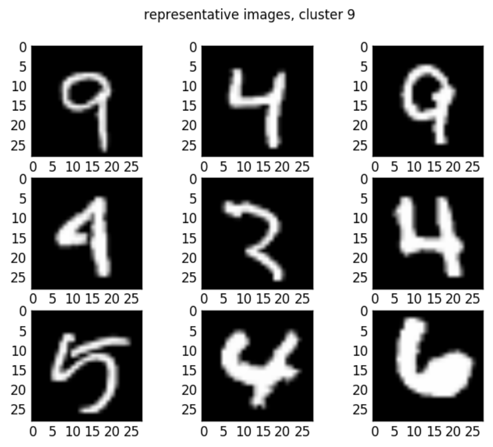


Figure 11: K=10, cluster 9 representative images

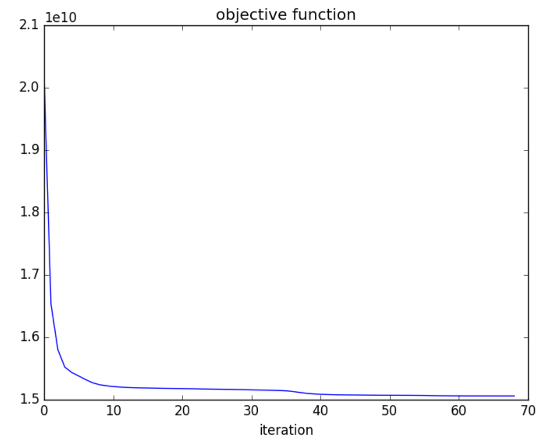


Figure 12: K=10, K-means objective function, regular K-means

Figures for K=6 simulation.

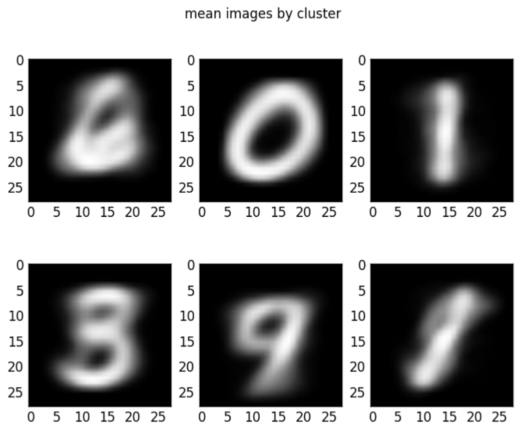


Figure 13: K=6, final mean images

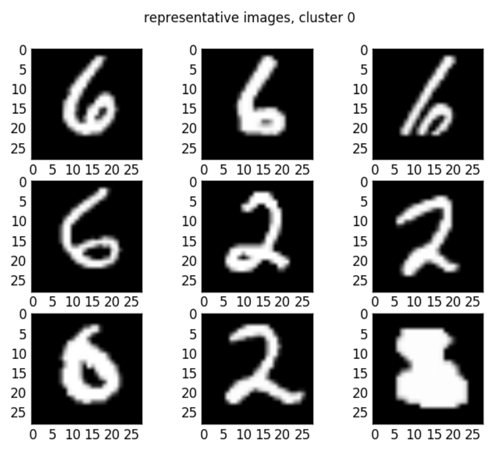


Figure 14: K=6, cluster 0 representative images

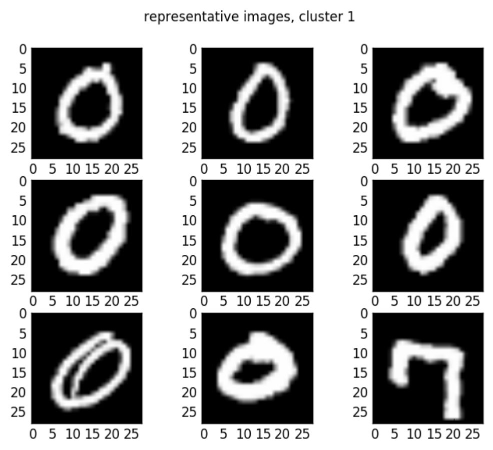


Figure 15: K=6, cluster 1 representative images

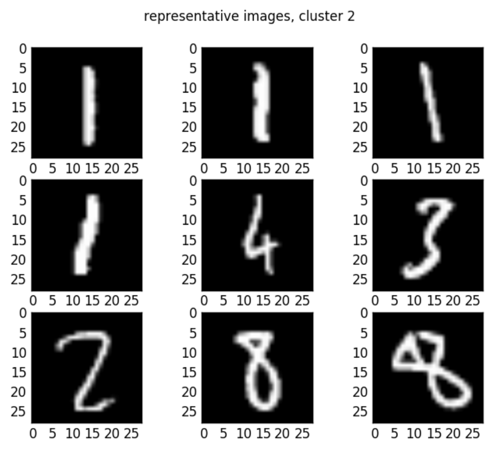


Figure 16: K=6, cluster 2 representative images

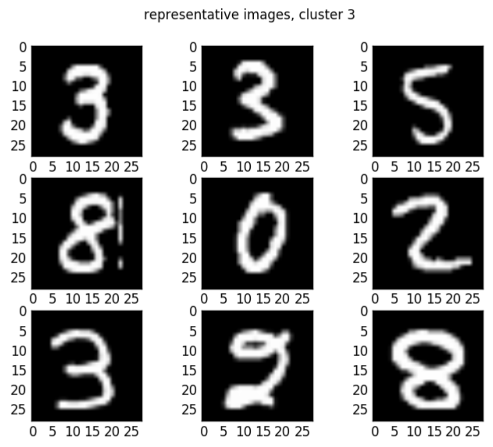


Figure 17: K=6, cluster 3 representative images

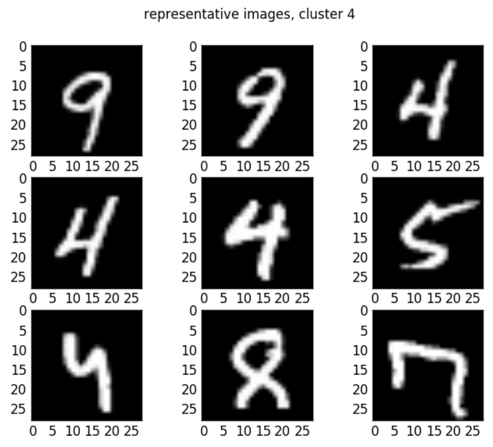


Figure 18: K=6, cluster 4 representative images

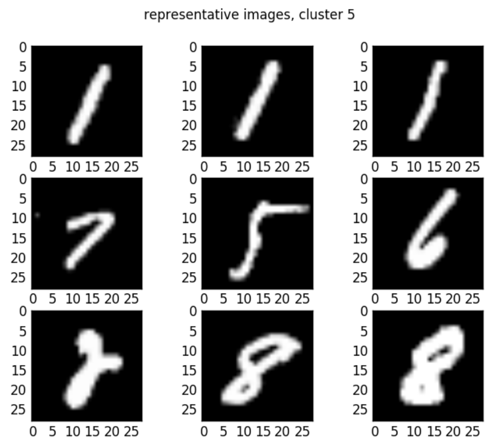


Figure 19: K=6, cluster 5 representative images

Figures for K=15 simulation.

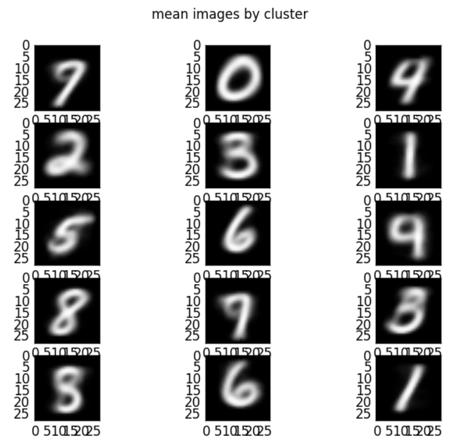


Figure 20: K=15, final mean images

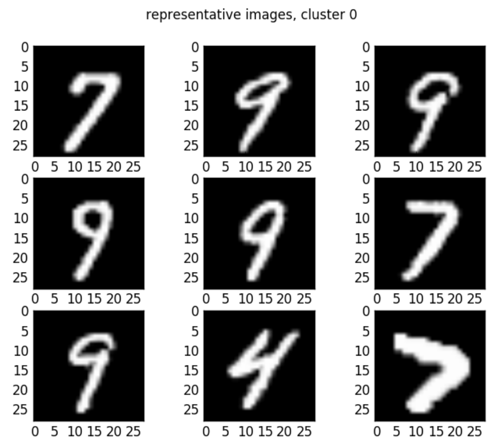


Figure 21: K=15, cluster 0 representative images

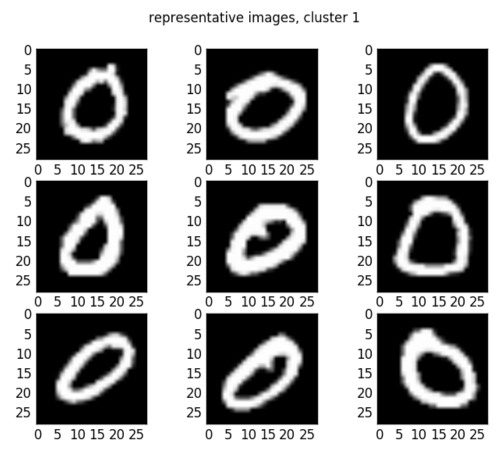


Figure 22: K=15, cluster 1 representative images

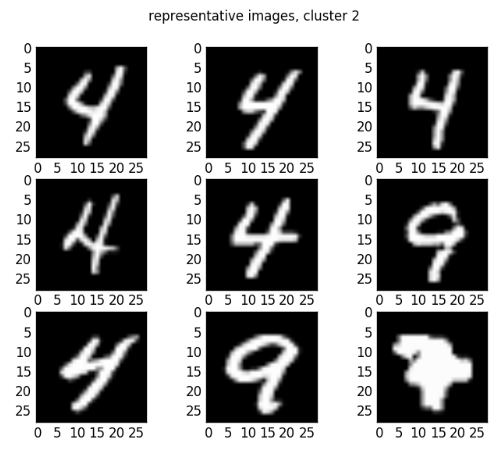


Figure 23: K=15, cluster 2 representative images

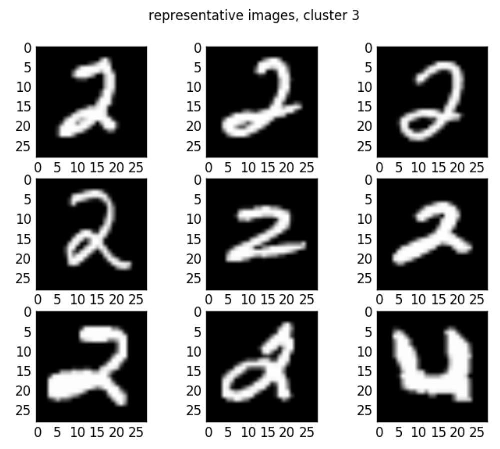


Figure 24: K=15, cluster 3 representative images

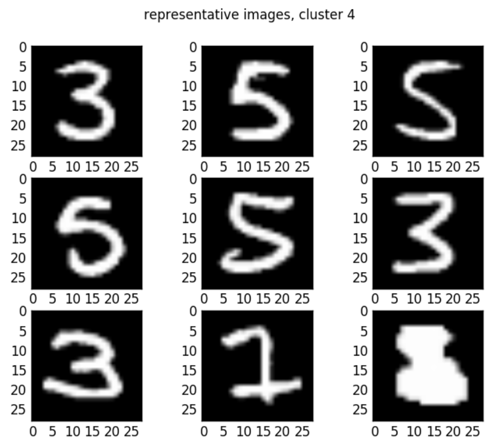


Figure 25: K=15, cluster 4 representative images

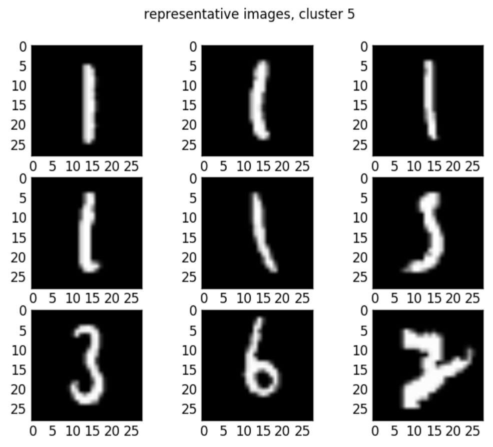


Figure 26: K=15, cluster 5 representative images

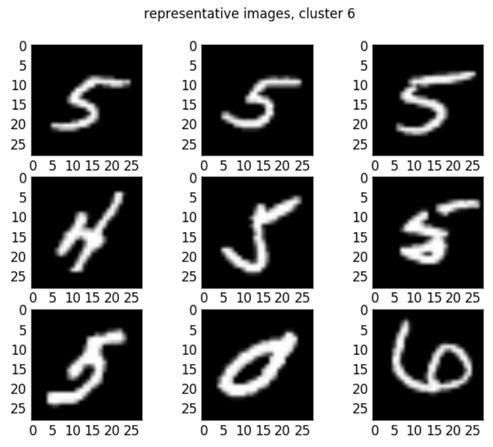


Figure 27: K=15, cluster 6 representative images

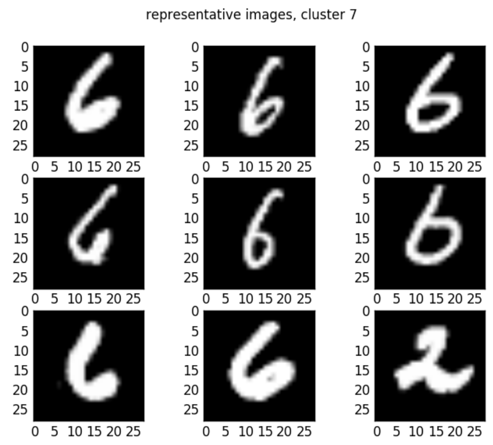


Figure 28: K=15, cluster 7 representative images

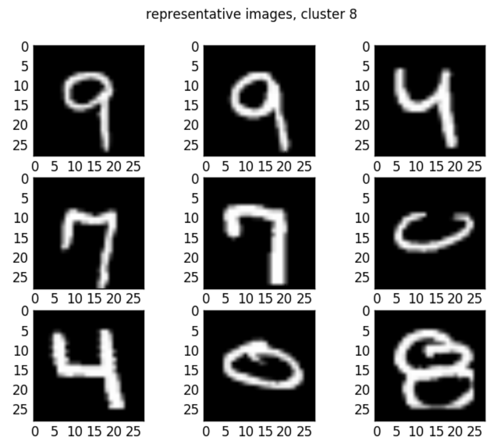


Figure 29: K=15, cluster 8 representative images

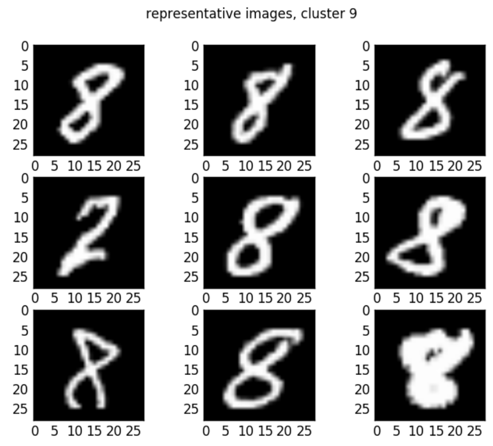


Figure 30: K=15, cluster 9 representative images

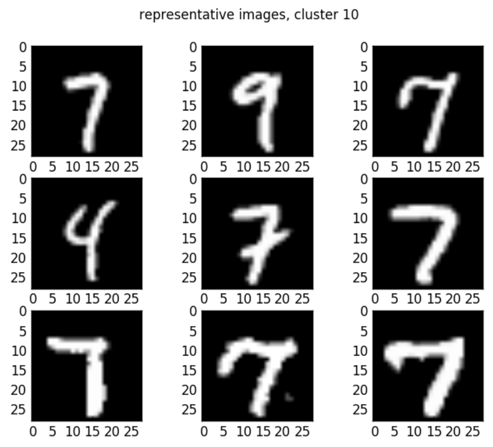


Figure 31: K=15, cluster 10 representative images

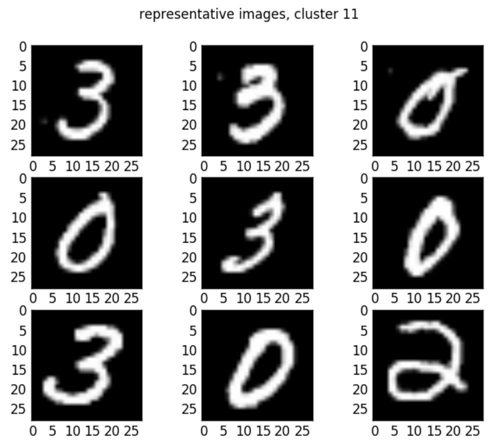


Figure 32: K=15, cluster 11 representative images

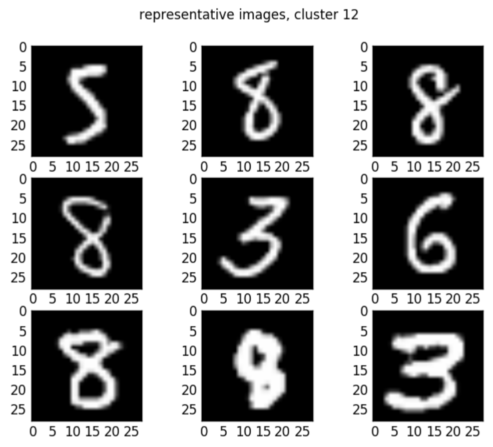


Figure 33: K=15, cluster 12 representative images

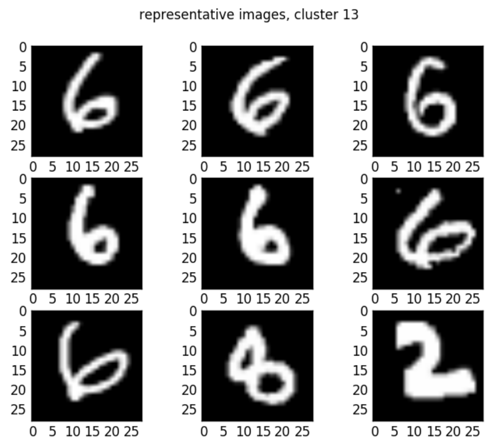


Figure 34: K=15, cluster 13 representative images

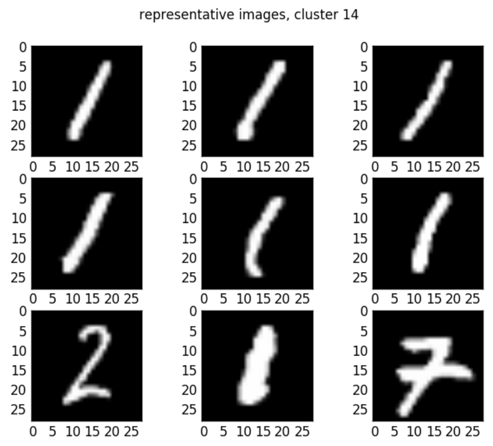


Figure 35: K=15, cluster 14 representative images

K-means++ plots.

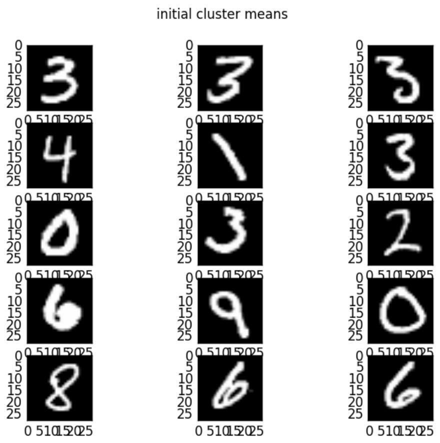


Figure 36: K=15, initial cluster means, K-means++

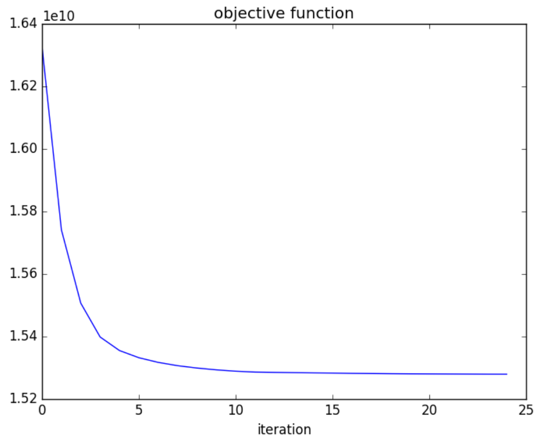


Figure 37: K=10, K-means objective function, K-means++

Problem 4 (Calibration, 1pt)
Approximately how long did this homework take you to complete? 30 hours.