Timothy Kang
tkang01@college.harvard.edu
CS181-S16

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

Collaborators: Jeff Chang

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

We begin with the volume of an n-ball, which was taken from the relevant Wikipedia page:

$$V_{n-\text{ball}} = \frac{\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}+1\right)}$$

The volume of the hypercube is trivial: $V_{\text{hypercube}} = 2^d$. From here, finding the fraction of the hypercube's volume that is contained within the hypersphere is simply a matter of dividing the volume of the n-ball by the volume of the hypersphere:

$$\text{fraction contained} = \frac{\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}+1\right)} \cdot \frac{1}{2^d}$$

Now we take the limit as $d \to \infty$:

$$\begin{aligned}
\lim_{d\to\infty} \frac{\pi^{\frac{d}{2}}}{\Gamma\left(\frac{d}{2}+1\right)} \cdot \frac{1}{2^d} &= \lim_{d\to\infty} \frac{\pi^{\frac{d}{2}}}{2^{\frac{d}{2}}} \cdot \frac{1}{2^{\frac{d}{2}}} \cdot \frac{1}{\Gamma\left(\frac{d}{2}+1\right)} \\
&= \lim_{d\to\infty} \frac{\left(\frac{\pi}{2}\right)^{\frac{d}{2}}}{2^{\frac{d}{2}}} \cdot \frac{1}{\Gamma\left(\frac{d}{2}+1\right)} \\
&\to 0
\end{aligned}$$

We know that the limit goes to 0 because $\frac{\pi}{2} \approx 1.57 < 2$, and the Gamma function is equivalent to a factorial. Therefore, the limit goes to 0.

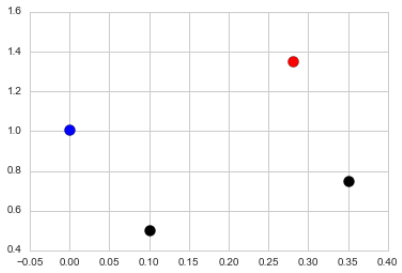
Intuitively, an n-ball with unit radius can be modeled as

$$x_1^2 + \cdots + x_n^2 = 1$$

As the number of dimensions increases, more and more of the x's will need to be closer to 0. With a hypercube, the corners are still at ± 1, regardless of how many dimensions, so the hypersphere captures less and less of the hypercube's volume as the dimensions increase.

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

WARMUP QUESTION

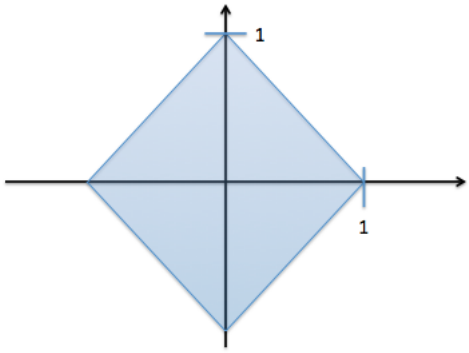
ℓ_1 **norm**: With this norm, the unit sphere is defined as

$$|x_1| + |x_2| = 1$$

This yields four possible equations:

$$x_2 = -x_1 - 1$$
$$x_2 = x + 1$$
$$x_2 = x_1 - 1$$
$$x_2 = 1 - x_1$$

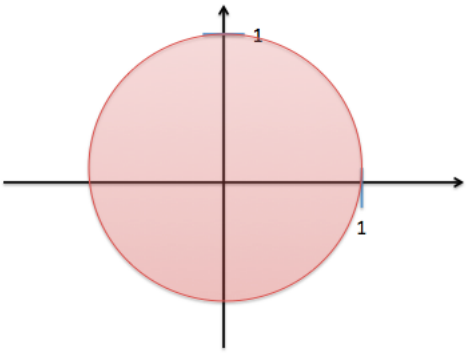
This produces the following unit sphere:



ℓ_2 **norm**: With this norm, the unit sphere is defined as

$$\sqrt{x_1^2 + x_2^2} = 1 \implies x_1^2 + x_2^2 = 1$$

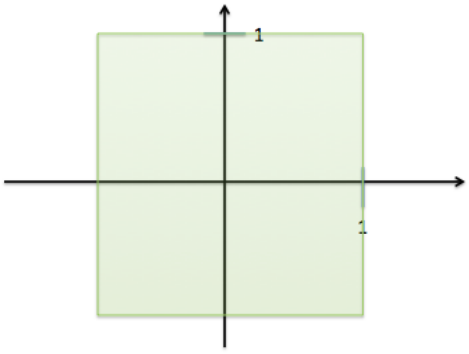
This is the familiar equation of a circle.



ℓ_3 **norm**: With this norm, the unit sphere is defined as

$$\max(|x_1|, |x_2|) = 1$$

This means that we have $x_1 = \pm 1$ or $x_2 = \pm 1$.



MAIN QUESTION

Before we continue, we define the left point in the black cluster (0.1,0.5) as A and the right point in the black cluster $(0.35, 0.75)$ as B. Also, for single-link, we look for the minimum distance within each cluster comparison (i.e. within Black-Blue); for complete-link, we look for the maximum distance; and for average-link, we just average all of the distances with each cluster comparison. Then, we return the minimum distance between all three cluster comparisons.

ℓ_1 **norm:** $\sum_i |x_i - y_i|$ (basically, add the differences between the x and y coordinates)

Inter-Cluster distances

- Black-Blue:
  - A: $|0.1 - 0| + |0.5 - 1.01| = 0.61$
  - B: $|0.35 - 0| + |0.75 - 1.01| = 0.61$
- Black-Red:
  - A: $|0.1 - 0.28| + |0.5 - 1.35| = 1.03$
  - B: $|0.35 - 0.28| + |0.75 - 1.35| = 0.67$
- Blue-Red: $|0 - 0.28| + |1.01 - 1.35| = 0.62$

Clustering methods

- Single link:
  - Black-Blue: 0.61
  - Black-Red: 0.67
  - Blue-Red: 0.62

  Therefore, Black-Blue merge first
- Complete link:
  - Black-Blue: 0.61
  - Black-Red: 1.03

  - Blue-Red: 0.62

  Therefore, Black-Blue merge first

- Average link:
  - Black-Blue: 0.61
  - Black-Red: 0.85
  - Blue-Red: 0.62

  Therefore, Black-Blue merge first

ℓ_2 **norm:** $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ (basically, the distance formula - all calculations were done using Wolfram alpha's distance formula calculator.)

Inter-Cluster distances

- Black-Blue:
  - A: 0.519
  - B: 0.436
- Black-Red:
  - A: 0.869
  - B: 0.604
- Red-Blue: 0.440

Clustering methods

- Single link:
  - Black-Blue: 0.436
  - Black-Red: 0.604
  - Blue-Red: 0.440

  Therefore, Black-blue merge first

- Complete link:
  - Black-Blue: 0.519
  - Black-Red: 0.869
  - Blue-Red: 0.440

  Therefore, Blue-Red merge first

- Average link:
  - Black-Blue: 0.4775
  - Black-Red: 0.7365
  - Blue-Red: 0.440

  Therefore, Blue-Red merge first

ℓ_∞ **norm:** This is the maximum of the absolute value of the difference between coordinates. Inter-Cluster distances

- Black-Blue:
  - A: 0.51
  - B: 0.35
- Black-Red:
  - A: 0.85
  - B: 0.60
- Red-Blue: 0.34

Clustering methods

- Single link:
  - Black-Blue: 0.35
  - Black-Red: 0.60
  - Blue-Red: 0.34

  Therefore, Blue-Red merge first
- Complete link:
  - Black-Blue: 0.51
  - Black-Red: 0.85
  - Blue-Red: 0.34

  Therefore, Blue-Red merge first
- Average link:
  - Black-Blue: 0.43
  - Black-Red: 0.725
  - Blue-Red: 0.34

  Therefore, Blue-Red merge first

	ℓ_1	ℓ_2	ℓ_∞
Single	Black-Blue	Black-Blue	Red-Blue
Complete	Black-Blue	Red-Blue	Red-Blue
Average	Black-Blue	Red-Blue	Red-Blue

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.

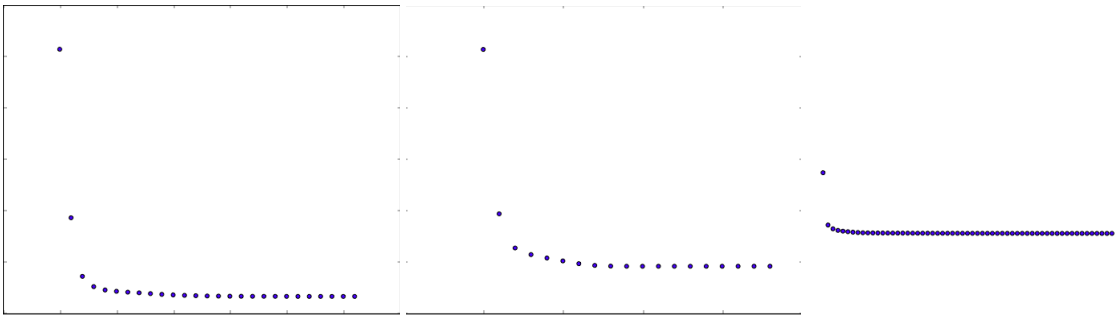
As in past problem sets, please include your plots in this document. There may be tons of plots for this problem, so feel free to take up multiple pages, as long as it is organized.

Solution

OBJECTIVE FUNCTION

$$J\left(\{\mathbf{r}_n\}_{n=1}^N, \{\mu_k\}_{k=1}^K\right) = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\|\mathbf{x}_n - \mu_k\|_2^2$$

(The μ should be boldfaced)



From left-to-right:

- $K = 10$ without KMeans++: 27 iterations and 76.39 seconds
- $K = 5$ without KMeans++: 19 iterations and 25.30 seconds

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

- $K = 10$ with KMeans++: 67 iterations and 109.02 seconds

It is fairly obvious that the K-means objective function, as a function of iteration, is monotonically decreasing in the number of iterations (up to a certain point).

RESULTS OF K-MEANS WITHOUT K-MEANS++
(Images are displayed from left-to-right)
Set 1: $K = 10$ - took 49 iterations and 139.18 seconds.
Means:

From left-to-right: 9. 5 or 3, 1, 6 or 2, 0, 3, 8, 9, 1, 7.
Representative Images:

Figure 1: 9

Figure 2: 5 or 3

Figure 3: 1

Figure 4: 6 or 2

Figure 5: 0

Figure 6: 3

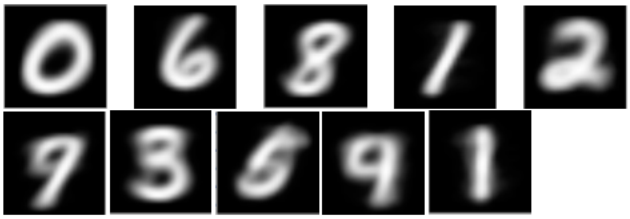
Figure 7: 8

Figure 8: 9

Figure 9: 1

Figure 10: 7

Set 2: $K = 10$ - took 42 iterations and 127.89 seconds.
Means:



From left-to-right: 0, 6, 8, 1, 2, 7 or 9, 3, 5, 9, 1.
Representative Images:

Figure 1: 0

Figure 2: 6

Figure 3: 8

Figure 4: 1

Figure 5: 2

Figure 6: 7 or 9

Figure 7: 3

Figure 8: 5

Figure 9: 9

Figure 10: 1

Between the two restarts (with same values of K), the results were largely the same. Both restarts took the same amount of time and produced, more-or-less, the same set of means. Also, the means were for the most part distinguishable: the images were not terribly blurry or merged (merged meaning overlapping numbers). It seems that K-means was not able to handle the numbers 4, 5, or 9 terribly well, as instances

of these numbers tended to be merged with other numbers (that is, I would often get a 4 overlapped with 9). Similarly, 1 appeared quite frequently, largely because its structure is noticeable in other numbers (i.e. 7 or 5). Finally, the representative images for each cluster in both restarts seemed to largely match with the means that we returned.

Set 3: $K = 3$ - took 52 iterations and 41.89 seconds.
Means:

From left-to-right: 9; 8,3,6, ???; ???.
Representative Images:

Figure 1: 9

Figure 2: 8, 3, 6, or ???

Figure 3: ???

Set 4: $K = 3$ - took 20 iterations and 17.37 seconds.
Means:

From left-to-right: ???, ???, 9 or 7.
Representative Images:

Figure 1: ???

Figure 2: ???

Figure 3: 9 or 7

I ran the script 3 more times with $K = 3$ and got mostly the same results for the other 3 trials. All of the restarts produced identical means and representative images. Furthermore, for the means of the clusters, all of them seem to be overlapped images of numbers, which makes sense since only having three clusters means that some of the numbers will have to be grouped together.

RESULTS OF K-MEANS WITH K-MEANS++
Set 5: $K = 10$ - took 37 iterations and 89.24 seconds.
Means:

From left-to-right: 8,2,6,0,3,7,8,9,7,1.
Representative Images:

Figure 1: 8

Figure 2: 2

Figure 3: 6

Figure 4: 0

Figure 5: 3

Figure 6: 9

Figure 7: 8

Figure 8: 4

Figure 9: 7

Figure 10: 1

After reading the `Arthur and Vassilvitskii` paper (http://theory.stanford.edu/ sergei/papers/kMeansPP-soda.pdf), I realized that the primary benefit of *K*-means++ initialization is the improved runtime. Specifically, according to both the paper and Wikipedia page, "With the k-means++ initialization, the algorithm is guaranteed to find a solution that is O(log k) competitive to the optimal k-means solution." While the initial selection process in *K*-means++ takes extra time, this allows the algorithm to converge to a quicker solution and reduce computation time.

To confirm this, I ran K-means with 10 clusters and K-means++ initialization four more times and got the following results: 23 iterations and 47.466 seconds, 37 iterations and 62.034 seconds, 25 iterations and 72.041 seconds, and 26 iterations and 80.323 seconds. Overall, the K-means++ initialization did, in fact, improve runtime, and the results were just as accurate as K-means with random initialization.

Problem 4 (Calibration, 1pt)

Approximately how long did this homework take you to complete?
Finishing it took like 5 hours. Writing it up took another 3.