ISYE 6740 Fall 2025
Homework 1 (100 points)
Will Pickard

In this homework, the superscript of a symbol x^i denotes the index of samples (not raising to ith power); this is a convention in this class. Please follow the homework submission instructions in the syllabus.

Provided Data: Questions marked with GS must be submitted to Gradescope. You must still include all your results and explanations in this PDF, and include your code in your submitted zip. Failure to pass all gradescope tests will result in losing 50% of the points for that question.

- (GS) Q3: Image compression using clustering [football.bmp, parrots.png]
- (GS) Q4: Political Blogs Spectral Clustering [nodes.txt, edges.txt]
- Q5: AI model compression [AImodel.npy]

All results that are present only in code/notebooks, but not in your PDF report will not be accepted for points. Handwritten answers will not receive credit.

1 Concept questions [25 points]

Please provide a brief answer to each question, and provide math arguments if asked.

1. (5 points) Consider a dataset with data points each having 3 features, e.g., $x^1 = \{\text{“Atlanta”}, \text{“house”}, 500k\}$, and $x^2 = \{\text{“Dallas”}, \text{“house”}, 300k\}$. Define a proper similarity function $d(x^i, x^j)$ for this kind of data, and argue why it is a reasonable choice. (Hint: The feature vector consists of categorial and real-valued features; for categorical variables, it is better to convert them into one-hot-keying binary vectors and use Hamming distance, and for real-valued features, you may use Euclidean distance, for instance. And then you can combine the similarity measure in some way.)

Solution 1

My similarity function for this dataset would be:

$$d(xi, xj) = d_{H_{norm}}(x^i_{categorical}, x^j_{categorical}) + d_{E_{norm}}(x^i_{numerical}, x^j_{numerical})$$

For the following reasons per step in the process:

(a) One-hot encoding the categorical features (features 1 and 2) and combining their binary vectors into one effectively moves these categorical variables into a numerical plane.

(b) Calculating the Hamming distance on these binary vectors results in a count of the different bits between the two vectors. So, higher distance means more dissimilarity in the categorical space.

(c) Calculate the euclidean distance for the real-valued feature (feature 3).

(d) Normalize the Hamming distance and euclidean distance calculations to a range of [0,1]. This is essential, since the euclidean distances would be far larger than the Hamming distances, meaning it would dominate the resulting similarity metric. Using the [0,1] range is chosen strictly because it is the standard.

This similarity function would properly handle mixed data and their distance metrics while preventing feature dominance. One might also add weighting to the calculation, but that would be dependent on use-case interests that are not in evidence.

2. (5 points) Show that the clustering assignment problem

$$\pi(i) = \arg\min_{j=1,\ldots,k} \|x^i - c^j\|^2$$

is equivalent to solving

$$\pi(i) = \arg\min_{j=1,\ldots,k} (c^j)^T \left(\frac{1}{2}c^j - x^i\right).$$

Note that the second approach will facilitate "vectorized" operation and be implemented in our demo code.

Solution 2

Proof. Beginning with the initial statement of equivalence

$$\arg\min_{j=1,\ldots,k} \|x^i - c^j\|^2 = \pi(i) = \arg\min_{j=1,\ldots,k} (c^j)^T \left(\frac{1}{2}c^j - x^i\right)$$

Now, we'll expand both sides

$$\arg\min_{j=1,\ldots,k} (x^i - c^j)^T(x^i - c^j) = \arg\min_{j=1,\ldots,k} \frac{1}{2}(c^j)^T c^j - (c^j)^T x^i$$
$$\arg\min_{j=1,\ldots,k} (x^i)^T(x^i) - (x^i)^T c^j - (c^j)^T x^i + (c^j)^T c^j =$$

Simplify with the basic property $a^T b = b^T a$

$$\arg\min_{j=1,\ldots,k} (x^i)^T(x^i) - 2(c^j)^T x^i + (c^j)^T c^j = \arg\min_{j=1,\ldots,k} \frac{1}{2}(c^j)^T c^j - (c^j)^T x^i$$

Now, the trickiest part of this proof is just dropping anything not related to j. Since the function is finding the minimum result given a series of indices j, any term unrelated to j is basically a constant. A constant value has no real impact on the outcome of the minimization.

As a basic example, if we were to find

$$\arg\min_{j=1,\ldots,k} 12 - b^j, \quad \text{given that} \quad b = [1,2,3,4]^T$$

the 12 constant really doesn't matter. The j that minimizes the function is simply the j corresponding to (in this case) the largest value in b. As such, we can basically remove the irrelevant term and end with:

$$\arg\min_{j=1,\ldots,k} -b^j, \quad \text{given that} \quad b = [1,2,3,4]^T$$

We can use this to remove irrelevant terms and even multiply entire sides of the equation by constants, since constants irrespective of j do not effect the outcome of the minimization. We will start by removing the $(x^i)^T x^i$ terms.

$$\arg\min_{j=1,\dots,k} -2(c^j)^T x^i + (c^j)^T c^j = \arg\min_{j=1,\dots,k} \frac{1}{2}(c^j)^T c^j - (c^j)^T x^i$$

Now multiply one side of the equation by a constant to prove equivalency.

$$\arg\min_{j=1,\dots,k} -2(c^j)^T x^i + (c^j)^T c^j = \arg\min_{j=1,\dots,k} 2(\frac{1}{2}(c^j)^T c^j - (c^j)^T x^i)$$
$$\arg\min_{j=1,\dots,k} (c^j)^T c^j - 2(c^j)^T x^i = \arg\min_{j=1,\dots,k} (c^j)^T c^j - 2(c^j)^T x^i)$$

□

3. (5 points) Why do different initializations for k-means lead to different results? In practice, to finding a better result facing this issue, what would you do?

Solution 3

Different initializations for k-means lead to different results because the outcome of k-means is heavily dependent on the randomized initial centroid selection. The iterative process of moving the random initial centroids to cluster means then reassigning the centroid if the WCSS is lower is not exhaustive. As such, it is prone to finding a local minimum in the minimization, but not the global minimum.

For instance, if two centroids end up collocated in an information-dense part of the data, they would likely partition a very small portion of the data. Using the image compression example, if two centroids are chosen with the same RGB values (let's say [255, 255, 255] for white as a common example), then one cluster would be empty and basically useless, even if there is much more information in the color space.

To solve this in practice, I would run the k-means algorithm many more times with different random initializations each time. This would give a better chance of finding the global minimum, or at least a lower local minimum than only running with one set of initializations. To the extreme, this could be run by exhaustively selecting every combination of k data points as initial centroids, which would certainly find the global minimum very very

4. (5 points) Why k-means will not have the issue of running a infinite number of iterations (suppose the stopping criterion is when the cluster assignments after another iteration do not change), in most settings?

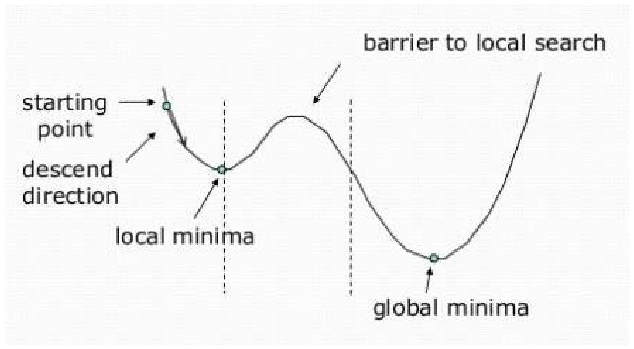


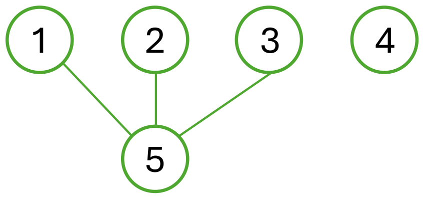
Figure 1: Local Minima vs. Global Minimum [CAO20]

Solution 4

The figure above graphically explains the basics of why k-means is unlikely to infinitely iterate. During the assignment step of k-means, each point is assigned to a centroid that is closest to it. This means that each point's distance to its centroid will either remain the same or decrease. During centroid updating, the centroid moving to the cluster's mean minimizes the WCSS, so the WCSS is guaranteed to either decrease or remain unchanged.

Since the WCSS can only get smaller, and sum-of-squares are guaranteed to be positive, this means that the objective function must converge to a local/global minimum or zero. Visually, a minimization on Figure 1 would end iteration after descending the graph to the local minimum.

5. (5 points) Consider the following simple graph



Write down the graph Laplacian matrix and find the eigenvectors associated with the zero eigenvalues. Explain how you find out the number of disconnected clusters in the graph and identify these disconnected clusters using these eigenvectors.

Solution 5

The adjacency and degree matrices for this graph are:

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix}$$

We get the Laplacian matrix $L = D - A$

$$L = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 \\ -1 & -1 & -1 & 0 & 3 \end{bmatrix}$$

We find the eigenvectors for the zero eigenvalues with $Lv = \lambda v = 0$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & -1 \\ -1 & -1 & -1 & 0 & 3 \end{bmatrix} \begin{bmatrix} v^1 \\ v^2 \\ v^3 \\ v^4 \\ v^5 \end{bmatrix} = 0$$

We can then use this python code to get the zero-eigenvalues and eigenvectors.

```
# This is Python code
L = np.array([
    [1, 0, 0, 0, -1],
    [0, 1, 0, 0, -1],
    [0, 0, 1, 0, -1],
    [0, 0, 0, 0, 0],
    [-1, -1, -1, 0, 3]
])

values, vectors = np.linalg.eig(L) # get eigendecomposition
```

$$eigenvalues = \begin{bmatrix} 0 \\ 4 \\ 1 \\ 0 \\ 1 \end{bmatrix}, eigenvectors = \begin{bmatrix} -0.2887 & -0.8165 & -0.5 & -0.1543 & 0 \\ -0.2887 & 0.4082 & -0.5 & 0.7715 & 0 \\ -0.2887 & 0.4082 & -0.5 & -0.6172 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0.8660 & 0 & -0.5 & 0 & 0 \end{bmatrix}$$

We can identify the number of disconnected clusters by determining the number of zero eigenvectors: 2. We can then identify the clusters by finding indicator vectors - vectors that have a constant value for all nodes in the cluster and zero for nodes out of the cluster. We can see two eigenvectors for this - $v = [-.5, -.5, -.5, 0, -.5]$ and $v = [0, 0, 0, 1, 0]$. These vectors show that node 4 is in a cluster alone, and the other cluster has nodes 1, 2, 3, and 5.

2 Math of k-means clustering [20 points]

Given m data points $\mathrm{x}^i \in \mathbb{R}^n$, $i = 1, \dots, m$, K-means clustering algorithm groups them into k clusters by minimizing the distortion function over $\{r^{ij}, \mu^j\}$

$$J = \frac{1}{mk} \sum_{i=1}^{m} \sum_{j=1}^{k} r^{ij} \|\mathrm{x}^i - \mu^j\|^2, \tag{1}$$

where $r^{ij} = 1$ if x^i belongs to the j-th cluster and $r^{ij} = 0$ otherwise.

1. (8 points) Derive mathematically that using the squared Euclidean distance $\|\mathrm{x}^i - \mu^j\|^2$ as the dissimilarity function, the centroid that minimizes the distortion function J for given assignments r^{ij} are given by
$$\mu^j = \frac{\sum_i r^{ij} \mathrm{x}^i}{\sum_i r^{ij}}.$$

That is, μ^j is the center of j-th cluster.
Hint: You may start by taking the partial derivative of J with respect to μ^j, with r^{ij} fixed.

Solution 2.1

To find the centroid that minimizes the distortion function, we can take the partial derivative of J with respect to μ^j.

$$\frac{\partial J}{\partial \mu^j} = \frac{1}{mk}\sum_{i=1}^{m}\sum_{j=1}^{k}[r^{ij}(x^i-\mu^j)^T(x^i-\mu^j)] \tag{2}$$

Focusing just on the partial derivative calculation:

$$\begin{aligned}\frac{\partial}{\partial \mu^j}[(x^i)^2-(\mu^j)^T(x^i)-(x^i)^T(\mu^j)+(\mu^j)^2] &= \frac{\partial}{\partial \mu^j}[(x^i)^2-2(\mu^j)^T(x^i)^{(1)}+(\mu^j)^2] && (3)\\ &= -2x^i+2\mu^j && (4)\\ &= -2(x^i-\mu^j) && (5)\end{aligned}$$

Plugging that partial derivative into (2):

$$\begin{aligned}\frac{\partial J}{\partial \mu^j} &= \frac{1}{mk}\sum_{i=1}^{m}[r^{ij}(-2(x^i-\mu^j))] && (6)\\ &= \frac{-2}{mk}\sum_{i=1}^{m}[r^{ij}x^i-r^{ij}\mu^j] && (7)\end{aligned}$$

Set this partial derivative equal to zero to accomplish the minimization with respect to μ^j:

$$\begin{aligned}0 &= \frac{-2}{mk}\sum_{i=1}^{m}[r^{ij}x^i-r^{ij}\mu^j] && (8)\\ &= \sum_{i=1}^{m}[r^{ij}x^i-r^{ij}\mu^j] && (9)\\ \mu^j\sum_{i=1}^{m}r^{ij} &= \sum_{i=1}^{m}r^{ij}x^i && (10)\\ \mu^j &= \frac{\sum_{i=1}^{m}r^{ij}x^i}{\sum_{i=1}^{m}r^{ij}} && (11)\end{aligned}$$

This confirms that the centroid that minimizes the distortion function for given assignments r^ij is the mean of the points in cluster j.

2. (7 points) Derive mathematically what should be the assignment variables r^{ij} be to minimize the distortion function J, when the centroids μ^j are fixed.

(1) The derivative of this term is a vector derivative identity $\frac{\partial(\mathbf{a}^T\mathbf{x})}{\partial \mathbf{x}} = \mathbf{a}$. See [PP08] equation 69.

2.1 Solution 2.2

This problem is more simply understood logically than written mathematically. The assignment variable r^{ij} can only be 0 or 1, and the sum of all r^{ij} for a given data point x^i must be 1. By finding the minimum distortion value when the centroids are fixed, we are trying to find the centroid μ^j that is closest to point x^i. This closest point will be where the L2 norm $\|x^i - \mu^j\|^2$ is smallest.

This can be shown more explicitly by expanding the summation for an index of i. This will give an example of finding the closest centroid to x^i.

$$\text{For } x^i = \hat{x} : \sum_{j=1}^{k} r^{ij}||x^i - \mu^j||^2 = r^{i1}||x^i - \mu^1||^2 + r^{i2}||x^i - \mu^2||^2 + \cdots + r^{ik}||x^i - \mu^k||^2$$

2.1.1 Conclusion and Mathematical Form

So, given $r^{ij} \in \{0, 1\}$ and $\sum_{j=1}^{k} r^{ij} = 1$, the mathematical representation of ”assign 1 to the centroid with the shortest distance to the point” is:

$$r^{ij} = \begin{cases} 1 & \text{if } j = \arg\min_j ||x^i - \mu^j||^2 \\ 0 & otherwise \end{cases}$$

3. (5 points) For the question above, now suppose we change the similar score to a “quadratic” distance (also known as Mahalanobis distance) for given and fixed positive definite matrix $\Sigma \in \mathbb{R}^{n\times n}$, and the distortion function becomes:

$$J = \frac{1}{mk} \sum_{i=1}^{m} \sum_{j=1}^{k} r^{ij} (\mathrm{x}^i - \mu^j)^T \Sigma (\mathrm{x}^i - \mu^j),$$

Derive what μ^j and r^{ij} becomes in this case.

2.2 Solution 2.3

2.2.1 Partial Derivative for μ^j

This will follow the same method as Solution 2.1 - take partial derivative with respect to μ^j and set it to zero

$$\frac{\partial J}{\partial \mu^j} = \frac{1}{mk} \sum_{i=1}^{m} r^{ij} \frac{\partial}{\partial \mu^j} (x^i - \mu^j)^T \Sigma (x^i - \mu^j) \tag{12}$$

We will use [PP08] equation 86, which is available since the TAs allowed it inthis Ed post.

$$\text{With rule 86 :} \quad \frac{\partial}{\partial s}(x - s)^T W (x - s) = -2W(x - s)$$

$$\frac{\partial J}{\partial \mu^j} = \frac{1}{mk} \sum_{i=1}^{m} r^{ij} (-2\Sigma(x^i - \mu^j)) \tag{13}$$

$$= \frac{-2}{mk} \sum_{i=1}^{m} r^{ij} \Sigma (x^i - \mu^j) \tag{14}$$

Setting equal to zero and solving:

$$0 = \frac{-2}{mk}\sum_{i=1}^{m} r^{ij}\Sigma(x^i - \mu^j) \tag{15}$$

$$\Sigma^{-1}(0) = \Sigma^{-1}\Sigma(\frac{-2}{mk}\sum_{i=1}^{m} r^{ij}(x^i - \mu^j))^{(2)} \tag{16}$$

$$0 = (\frac{-2}{mk}\sum_{i=1}^{m} r^{ij}(x^i - \mu^j))^{(3)} \tag{17}$$

From here, we see that the derivative is equivalent to Solution 2.1 equation 6, so the answer will be the same.

$$\mu^j = \frac{\sum_{i=1}^{m} r^{ij}x^i}{\sum_{i=1}^{m} r^{ij}}$$

2.2.2 Assignment variables r^{ij}

Similar to above, the answer to this question is going to be the same as the answer in Solution 2.2. Since Σ is constant, it has the same impact on the distance calculation for every combination of x^i and μ^j. The correct r^{ij} will again be the 1 assignment where the distance metric smallest, and zero assignment everywhere else.

$$r^{ij} = \begin{cases} 1 & \text{if } j = \arg\min_j (x^i - \mu^j)^T\Sigma(x^i - \mu^j) \\ 0 & \textit{otherwise} \end{cases}$$

3 Image compression using clustering [20 points]

In this programming assignment, you are going to apply clustering algorithms for image compression. This can also be viewed as an example of segmenting colors in an automated fashion using K-means clustering.

Your task is to implement K-*means* for this purpose. **It is required you implement the algorithms yourself rather than calling k-means from a package. However, it is ok to use standard packages for supplementary tasks, e.g., file i/o, linear algebra, distance calculations, and visualization.**

Formatting instruction

As a starting point, we suggest the following input/output signature for your k-means algorithm.

Input

- `pixels`: the input image representation. Each row contains one data point (pixel). For image dataset, it contains 3 columns, each column corresponding to Red, Green, and Blue components. Each component has an integer value between 0 and 255.
- `k`: the number of desired clusters.

Output

- `class`: cluster assignment of each data point in pixels. The assignment should be 1, 2, 3, etc. For $k = 5$, for example, each cell of the class should be either 1, 2, 3, 4, or 5. The output should be a column vector with `size(pixels, 1)` elements.

(2) We can move Σ out of the summation because it is not reliant on the i component, and matrix multiplication over vectors allows it. Since Σ is positive definite, it can be inverted to Σ^{-1} per [PP08] 9.6.4.

(3) $\Sigma^{-1}\Sigma = I$ per [PP08] equation 145, which effectively makes it a multiplication by 1 for this summation.

- `centroid`: location of k centroids (or representatives) in your result. With images, each centroid corresponds to the representative color of each cluster. The output should be a matrix with K rows and 3 columns. The range of values should be [0, 255], possibly floating point numbers.

Hand-in

Both your code and report will be evaluated. You must pass the gradescope unit tests to receive full credit, additionally upload your code into canvas and include your results in your PDF report separate from your code zip file. In your report, answer the following questions:

1. (5 points) Use k-means with squared-ℓ_2 norm as a metric for `parrots.png` and `football.bmp` and also choose a third picture of your own to work on. Your chosen image should meet the following requirements:
    - Full Color (No black and white images)
    - Recommended size between 200x150 and 400x300. Larger images are acceptable assuming they meet runtime requirements listed below in the Note, Smaller images are not acceptable.

    Run your k-means implementation with these pictures, with several different $k = 3, 6, 12, 24, 48$.

    Comment: Your algorithm will segment the image into k regions in the RGB color space. For each pixel in the input image, the algorithm returns a label corresponding to a cluster.

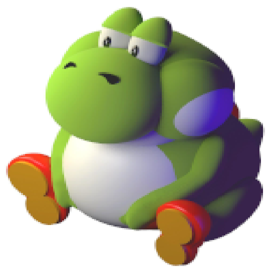
    Run your k-means implementation (with squared-ℓ_2 norm) with random initialization centroids. Due to the nature of randomness, Please try multiple times and report the only the best seed (in terms of image quality). Please provide the following deliverables:
    - For every K on all 3 images, the reconstructed image
    - The time in seconds it takes to converge for every K on each image
    - The number of iterations to convergence for every K on each image

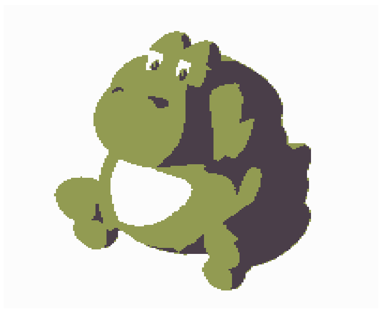
3.1 K-Means with squared-ℓ_2 Norm

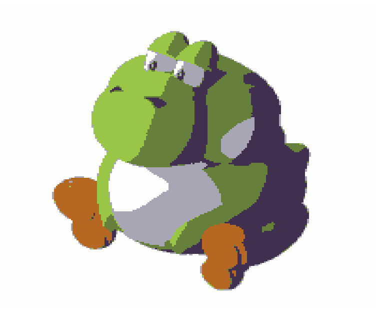
My results for the squared-ℓ_2 Norm were encouraging. I did manage to vectorize the distance calculation, which allowed my program to run in time. After testing multiple random seeds, I found a couple that worked quite well. I did have trouble with duplicate centroids in my chosen image, since a lot of the image is white. This lead to centroid initialization choosing the same values, and I got around that by changing the centroid initialization to pick from a set of unique values.

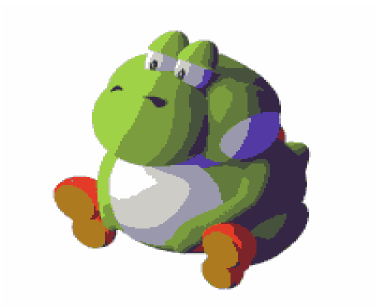
3.1.1 Chosen Image

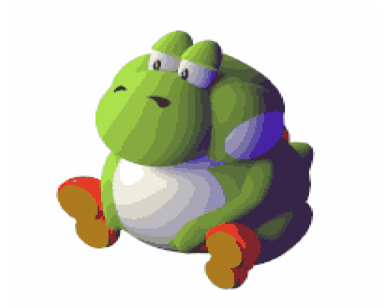
The results from my chosen image were good. Obviously, k=3 is a poor approximation, but the later cluster numbers do a good job of representing the image. However, the compression certainly struggles with the shadow gradient in the image.


(a) Original Image


(b) k=3 |i=8 |t=0.0415


(c) k=6 |i=20 |t=0.0931


(d) k=12 |i=21 |t=0.1260


(e) k=24 |i=53 |t=0.4679

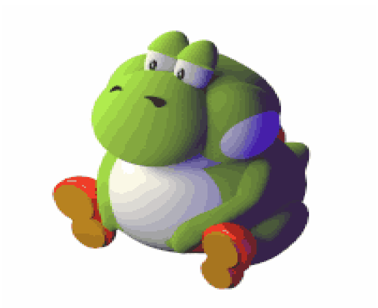
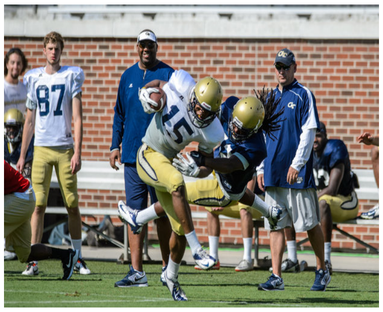
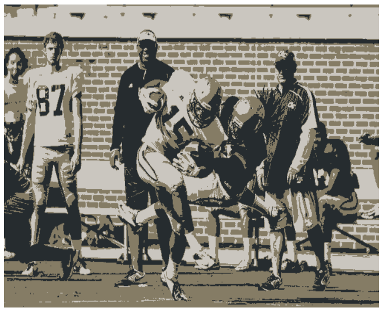

(f) k=48 |i=53 |t=0.51

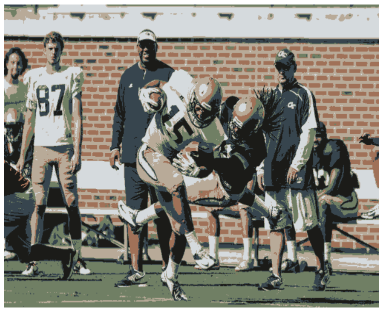
Figure 2: L2 Results on chosen image

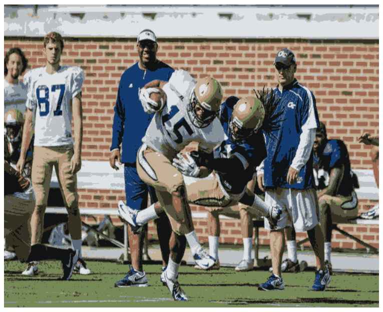
3.1.2 Football Image

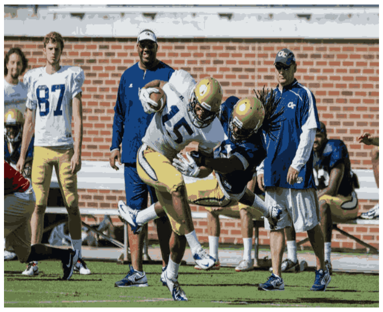
The replication of the football image is the most promising compression in my view. The k=48 result looks very similar to the original image at a glance.


(a) Original Image


(b) k=3 |i=14 |t=0.3505


(c) k=6 |i=17 |t=0.4484


(d) k=12 |i=44 |t=1.2823


(e) k=24 |i=36 |t=1.6508

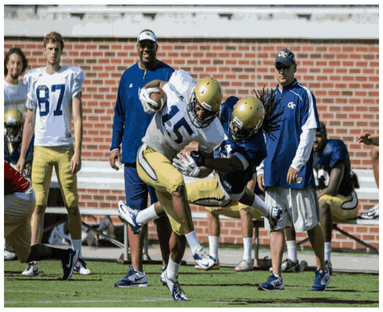
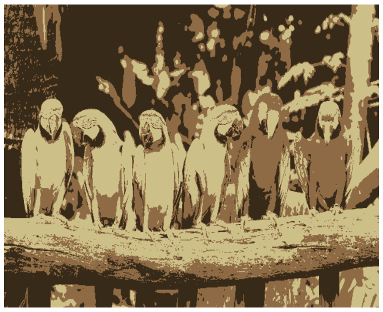

(f) k=48 |i=61 |t=4.5763

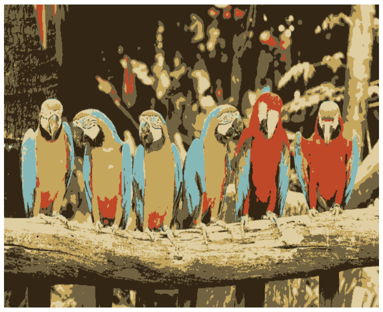
Figure 3: L2 Results on Football image

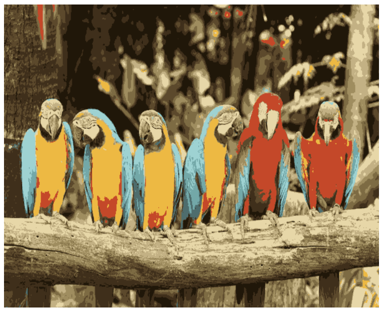
3.1.3 Parrots

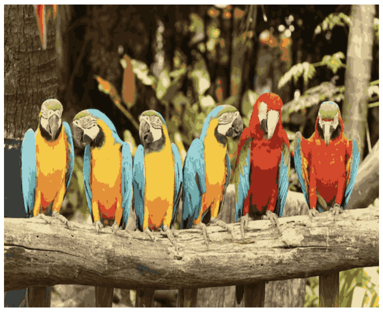
The parrots image certainly loses out on compression due to the vibrant and many colors in the original image.

(a) Original Image


(b) k=3 |i=7 |t=0.2496


(c) k=6 |i=24 |t=0.6373


(d) k=12 |i=43 |t=1.4784


(e) k=24 |i=49 |t=2.7014

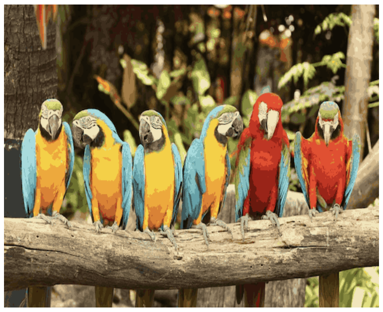

(f) k=48 |i=204 |t=14.2662

Figure 4: L2 Results on Parrots image

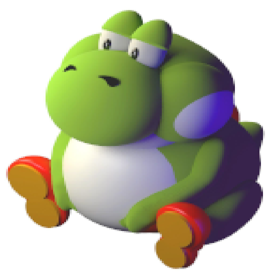
2. (10 points) Now adjust your k-means implementation to use the Manhattan distance (ℓ_1 norm) metric. Please provide all of the same above deliverables for all three images. Provide a comment on any differences you see in the results between using the L2 norm and L1 norm metrics.

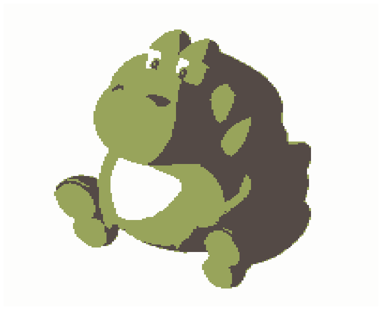
3.2 K-means with $_1$ Norm

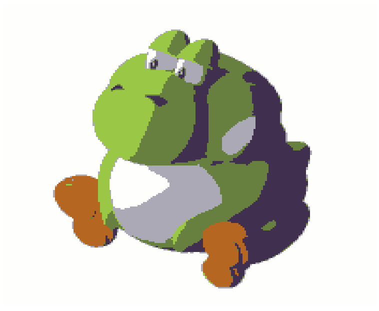
The L1 norm results were very similar to the L2 norm results. If anything, I think that L1 performs slightly worse at very low k values, which is more stark in the parrots and chosen images.

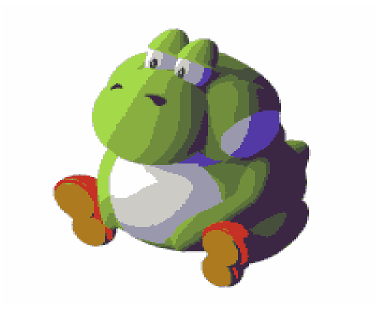
I also did not vectorize the L1 norm distance calculation, so the times were significantly higher.

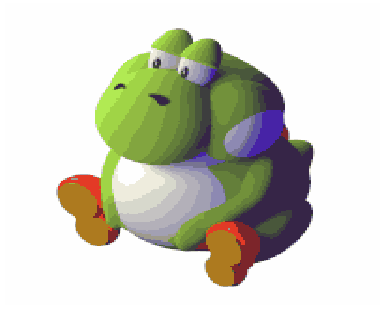
3.2.1 Chosen Image


(a) Original Image


(b) k=3 |i=7 |t=0.0503


(c) k=6 |i=17 |t=0.1428


(d) k=12 |i=17 |t=0.2351


(e) k=24 |i=36 |t=0.8384

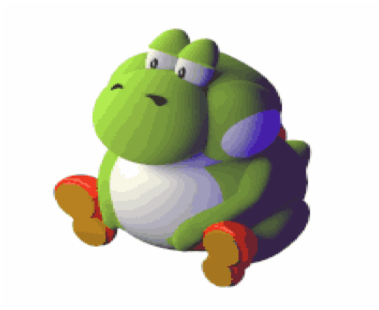
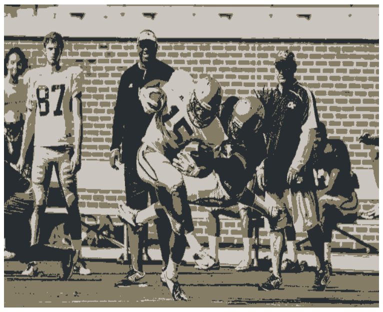
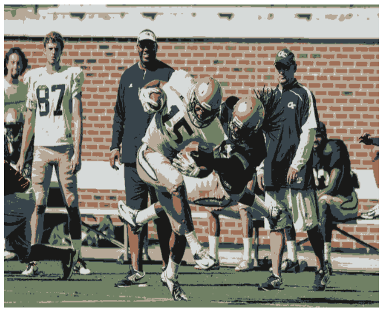

(f) k=48 |i=71 |t=3.1735

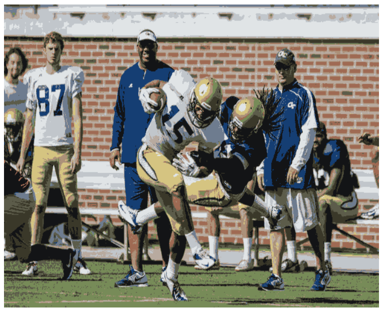
Figure 5: L1 Results on chosen image

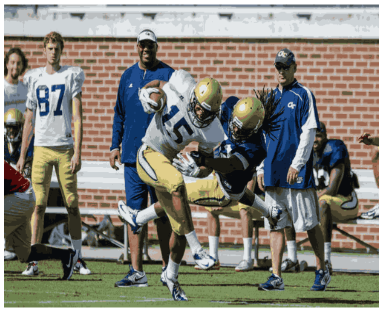
3.2.2 Football Image

(a) Original Image


(b) k=3 |i=13 |t=0.5001


(c) k=6 |i=13 |t=0.7170


(d) k=12 |i=30 |t=2.5241


(e) k=24 |i=62 |t=9.3703

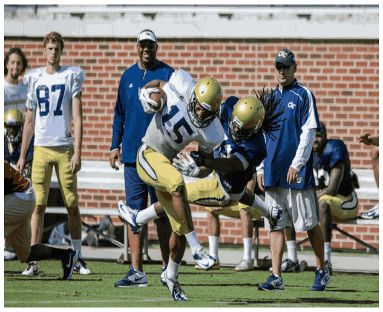
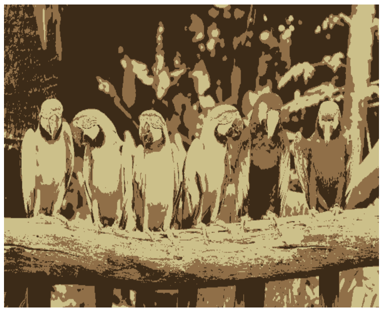
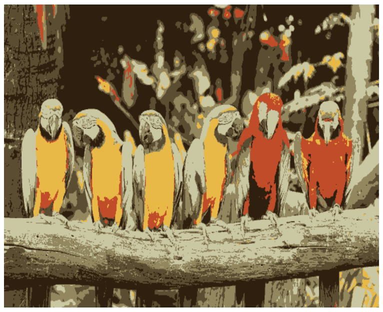

(f) k=48 |i=56 |t=16.2468

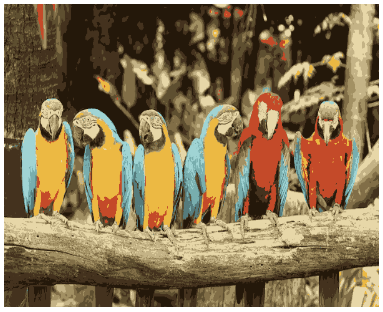
Figure 6: L1 Results on Football image

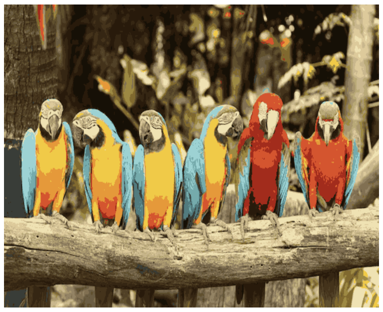
3.2.3 Parrots

(a) Original Image


(b) k=3 |i=11 |t=0.4390


(c) k=6 |i=27 |t=1.3320


(d) k=12 |i=32 |t=2.6719


(e) k=24 |i=26 |t=4.2231

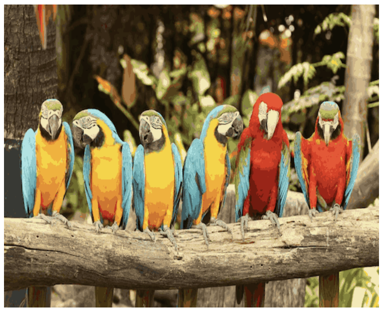

(f) k=48 |i=125 |t=37.0554

Figure 7: L1 Results on Parrots image

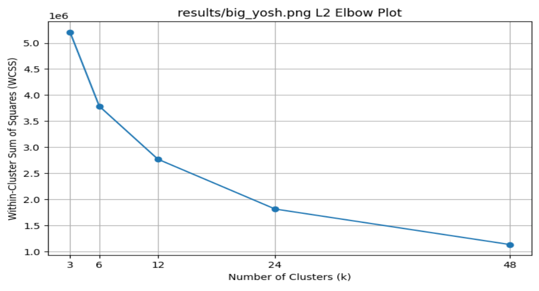
3. (5 points) Describe a method to find the best k. What is your best k?

3.3 Finding Best K

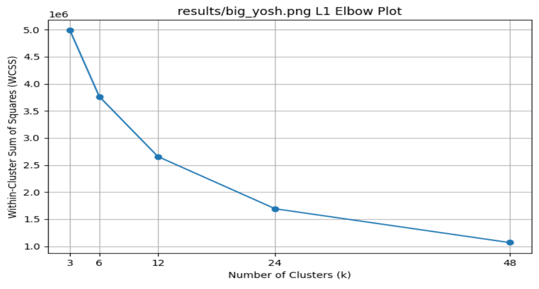
To find the best K, I chose to graph elbow plots of the within-cluster-sum-of-squares for each k value. This calculates the sum of squared differences between the centroid in each converged cluster to each point in that cluster. A lower WCSS indicates a more tightly-packed cluster. A lower WCSS for the same image should indicate a better clustering.

When viewing the elbow plots, the best k can be chosen by picking the k value for which the WCSS measures start having relatively smaller differences for increasing k. I think the best overall k for all these images would be 48. Though, for some it could be smaller and trialing more k values would help determine which is closer.

3.3.1 Chosen Image

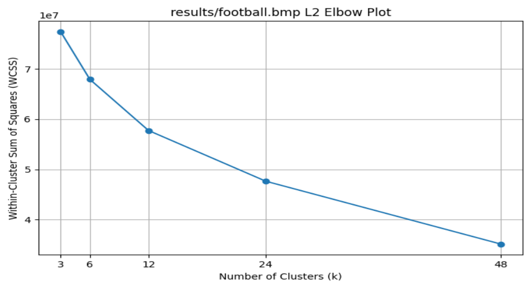


(a) Chosen Image L2 - best k = 24

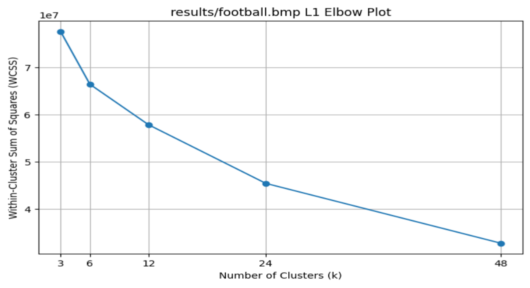


(b) Chosen Image L1 - best k = 24

3.3.2 Football Image

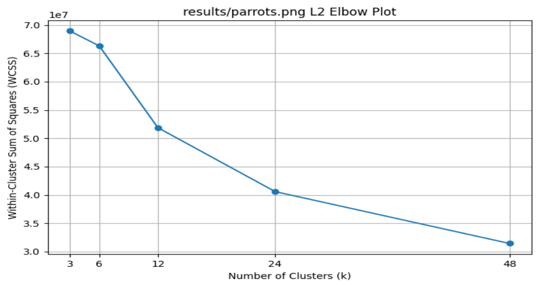


(a) Football Image L2 - best k = 48

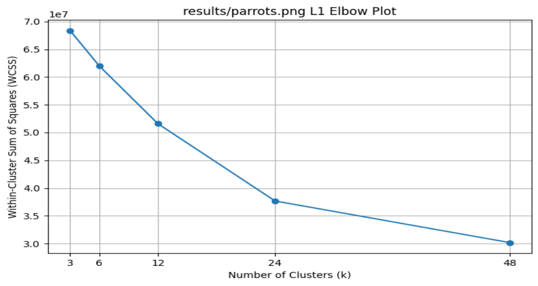


(b) Football Image L1 - best k = 48

3.3.3 Parrots Image



(a) Parrots Image L2 - best k = 24



(b) Parrots Image L1 - best k = 24

4 Political blogs dataset [20 points]

We will study a political blog dataset first compiled for the paper Lada A. Adamic and Natalie Glance, "The political blogosphere and the 2004 US Election", in Proceedings of the WWW-2005 Workshop on the Weblogging Ecosystem (2005). It is assumed that blog-site with the same political orientation are more likely to link to each other, thus, forming a "community" or "cluster" in a graph. In this question, we will see whether or not this hypothesis is likely to be true based on the data.

- The dataset `nodes.txt` contains a graph with $n = 1490$ vertices ("nodes") corresponding to political blogs.
- The dataset `edges.txt` contains edges between the vertices. You may remove isolated nodes (nodes that are not connected to any other nodes) in the pre-processing.

We will treat the network as an undirected graph; thus, when constructing the adjacency matrix, make it symmetrical by, e.g., set the entry in the adjacency matrix to be one whether there is an edge between the two nodes (in either direction).

In addition, each vertex has a 0-1 label (in the 3rd column of the data file) corresponding to the true political orientation of that blog. We will consider this as the true label and check whether spectral clustering will cluster nodes with the same political orientation as possible.

1. (10 points) Use spectral clustering to find the $k = 2, 5, 10, 30, 50$ clusters in the network of political blogs (each node is a blog, and their edges are defined in the file `edges.txt`). Find majority labels in each cluster for different k values, respectively. For example, if there are $k = 2$ clusters, and their labels are $\{0, 1, 1, 1\}$ and $\{0, 0, 1\}$ then the majority label for the first cluster is 1 and for the second cluster is 0. **It is required you implement the algorithms yourself rather than calling from a package.**

   Now compare the majority label with the individual labels in each cluster, and report the *mismatch* *rate* for each cluster, when $k = 2, 5, 10, 30, 50$. For instance, in the example above, the mismatch rate for the first cluster is 1/4 (only the first node differs from the majority), and the second cluster is 1/3.

Spectral Clustering Results

The following table contains the mismatch rates per cluster and average mismatch rate per k. I found that k=5 has the lowest average mismatch rate, with the mismatch rate notably increasing when partitioning the graph into more clusters.

		Number Clusters				
		2	5	10	30	50
Cluster ID	1	47.95%	23.17%	1.92%	2.33%	2.22%
	2	0.00%	2.33%	2.23%	0.00%	2.90%
	3		12.28%	2.86%	0.00%	2.17%
	4		30.00%	13.04%	2.50%	2.70%
	5		2.32%	45.83%	1.49%	5.00%
	6			20.00%	1.06%	0.00%
	7			2.82%	7.69%	4.55%
	8			12.50%	5.13%	0.00%
	9			0.00%	17.50%	0.00%
	10			42.11%	0.00%	29.73%
	11				0.00%	0.00%
	12				0.00%	0.00%
	13				47.22%	0.00%
	14				1.92%	14.29%
	15				21.95%	18.75%
	16				9.38%	22.22%
	17				9.09%	0.00%
	18				6.67%	3.57%
	19				0.00%	10.34%
	20				16.67%	0.00%
	21				3.33%	7.14%
	22				4.00%	9.38%
	23				5.00%	8.33%
	24				3.28%	4.88%
	25				6.41%	0.00%
	26				0.00%	0.00%
	27				0.00%	8.33%
	28				25.00%	9.76%
	29				0.00%	0.00%
	30				0.00%	4.35%
	31					0.00%
	32					6.06%
	33					9.09%
	34					11.54%
	35					14.29%
	36					0.00%
	37					45.83%
	38					0.00%
	39					0.00%
	40					0.00%
	41					0.00%
	42					4.17%
	43					8.33%
	44					0.00%
	45					20.00%
	46					0.00%
	47					23.08%
	48					0.00%
	49					0.00%
	50					14.29%
	Total	47.88%	4.41%	5.72%	5.96%	6.45%

2. (10 points) Tune your k and find the number of clusters to achieve a reasonably small *mismatch rate*. Please explain how you tune k and what is the achieved mismatch rate. Please explain intuitively what this result tells about the network community structure.

Spectral Clustering Conclusions

To tune k, I identified the most promising range of k values and tested different k values more granularly in that range. Since k=5 had the best results, and the error increased with the high k value 50, I decided to test k=[2,19]. I mostly wanted to see if there was a better k between 2 and 10, since that's the unknown range with 5 in it. However, testing up to 19 also allowed me to see if the upward error rate trend was stable.

In the end, 5 turned out to be the best k value and the trend was decently stable. This tells me that the political blog map is naturally connected in about 5 groups.

Cluster	Average Mismatch Rate
2	0.48%
3	0.48%
4	0.05%
5	0.04%
6	0.05%
7	0.05%
8	0.05%
9	0.05%
10	0.06%
11	0.05%
12	0.06%
13	0.06%
14	0.06%
15	0.06%
16	0.06%
17	0.06%
18	0.07%
19	0.06%

5 Compression of AI model using k-means [15 points]

This question ties clustering directly to vector quantization, which is a widely used real-world trick in compressing embedding-heavy ML models (e.g., recommender systems, NLP, vision).

You are working on deploying a large-scale recommendation system. The model contains a massive embedding table for users and items, with millions of vectors. Storing all embeddings in 32-bit floating point format would require several hundred gigabytes — too large for production deployment. A popular approach to compress embedding tables is **k-means vector quantization**, where embeddings are clustered and only the cluster centroids plus cluster assignments are stored. You can use the Kmeans package from 'sklearn.cluster' and numpy for this problem.

Tasks

1. (5 points) **Data Preparation:** You are provided with a dataset `AImodel.npy` of synthetic embeddings which can be read with numpy. Normalize the embeddings to unit length.

Run k-means clustering with $k = 256$. Instead of storing each embedding directly, you will now store:

- A centroid table of shape $[256, d]$
- An index (0–255) for each embedding

Provide an explanation why normalization may be important to perform before clustering for compression. Additionally, compute and report the total memory usage in MB to 4 decimal places before and after the compression. Memory usage can be calculated as a combination of your embedding bytes and your index bytes. (assume float32 Embeddings = 4 bytes, index stored as uint8 = 1 byte).

5.1 Data Preparation Solution

Normalization in this case can be important to perform before clustering for compression with k-means because k-means is very sensitive to large magnitudes. For compression, we mainly want to focus on the direction of the matrix transformation instead of the magnitude of them.

The original embedding table is a 1000x32 table of float32 values. The size of the original embeddings in memory is:

$$32 * 1000 \text{ float32 embeddings} * 4 \text{ bytes} = 0.128 \text{ MB}$$

After compressing with k=256, I store the list of centroids in a 256x32 array of float32 values and a 1000x1 array of uint8 indices. When reconstructing, the indices array guides which of the 256 centroid values is used to approximate the original embeddings and recreate the 1000x32 table. However the memory compression benefit is huge, as storing the compressed version severely reduces the number of float32 values in memory:

$$(32 * 256 \text{ float32 embeddings} * 4 \text{ bytes }) + 1000 \text{ bytes of uint8 indices} = 0.0338 \text{ MB}$$

2. (5 points) **Reconstruction Error:**

Given the index assignments produced by k-means, replace each embedding vector with its corresponding centroid. Define the cosine similarity between vectors a and b as:

$$\text{cosine\_similarity}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}|_2, |\mathbf{b}|_2}$$

where $\cdot$ denotes the dot product and $||_2$ denotes the Euclidean (L2) norm. Compute and report the average cosine similarity between all pairs of original and reconstructed embeddings:

$$\frac{1}{n} \sum_{i=1}^{n} \text{cosine\_similarity}(\mathbf{x}_i, \hat{\mathbf{x}}_i)$$

where $\mathbf{x}_i$ is the original embedding and $\hat{\mathbf{x}}_i$ is its quantized (centroid) version.

Hint: In NumPy, this can be computed as the dot product divided by the product of norms for each pair of vectors.

Interpret whether the compression preserves semantic similarity well, and explain what your specific calculated average cosine similarity value represents in the context of model compression.

5.2 Reconstruction Error

I calculated the cosine similarity using the vectorized form. To do this, I first had to reconstruct the 1000x32 quantized embedding table from the k-means result. The average cosine similarity between the original vectors and each quantized vector was 0.9242. The compression ratio with k=256 was $\frac{\text{original memory usage}}{\text{compressed memory usage}} = 3.787$. This means that the k-means vector compression was 3.787 times more memory efficient at the cost of a small degree of loss in the directional fidelity of the original model vectors.

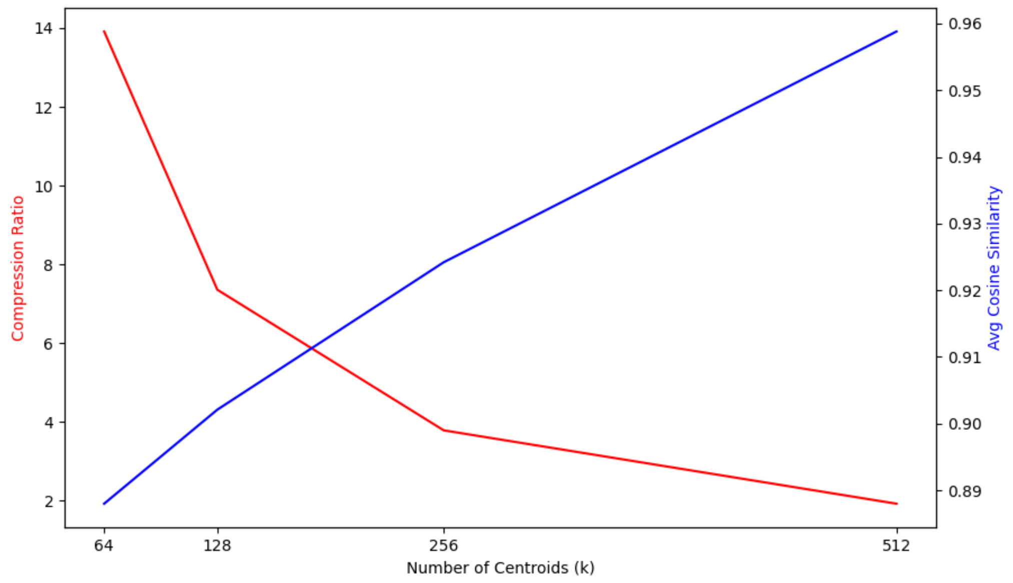


Figure 11: Compression Ratio and Reconstruction Quality vs K

3. (5 points) **Model Selection & Discussion:** Try different values of k (64, 128, 256, 512) and compare compression ratio vs. reconstruction quality. Provide plots showing how the compression ratio and the reconstruction quality changes with K. Additionally, Discuss the trade-off between model size and accuracy.

5.3 Model Selection and Discussion

The figure above shows my results for the compression at different values of k. There is a clear tradeoff in cosine similarity and compression ratio. The fewer centroids, the fewer float32 values need to be stored, which is a major boost to compression. However, fewer centroids to represent the original vectors also reduces the cosine similarity. How many centroids is the correct number is based on how much memory efficiency is required and how much destruction of model integrity is tolerable. In this case, I find that a 14x compression ratio at k=64 for a 0.90 cosine similarity to be a pretty compelling reason to use 64 centroids, but it is dependent on the situation.

References

[PP08] K. B. Petersen and M. S. Pedersen. *The Matrix Cookbook*. Version 20081110. Oct. 2008. URL: http://www2.imm.dtu.dk/pubdb/p.php?3274.

[CAO20] Seval Capraz, Halil Azyıkmış, and Adnan Ozsoy. "An Optimized GPU-Accelerated Route Planning of Multi-UAV Systems Using Simulated Annealing". In: *International Journal of Machine Learning and Computing* 10 (May 2020). DOI: 10.18178/ijmlc.2020.10.3.959.