ASSIGNMENT 3

October 16, 2019

Bar Kadosh (bk497), Ben Kadosh (bk499)

CS5785: Applied Machine Learning

Instructor: Nathan Kallus, Teaching Assistants: Xiaojie Mao, Yichun Hu

Contents

0.1 Summary

Our general approach to the coding sections of this assignment was to cleanse and transform the data as needed to make it easy to use and provide the desired insights. Below we describe the approach in more detail for each problem.

For exercise 1 with eigenfaces, we used the code provided to load in the training and test data in the desired format and plotted the desired images. We leveraged the numpy library to calculate the mean of the training set (average face μ) and to find the difference between it and each feature vector in the set.

We then performed SVD on the training data X, using numpy's np.linalg.svd and stored the components U, Σ, and V^T. Once we had the SVD components we approximated X using portions of the components U, Σ, and V^T up to a given r. Following that step, we created r-dimensional feature vectors in r-dimensional feature matrices F and F_{test}. Lastly, we used logistic regression and trained a model on the r-dimensional matrix F and the training labels to classify F_{test}. Our accuracy results are plotted below along with further details for each step in the exercise.

For exercise 2, we were tasked with using K-means to cluster data in two different formats - words by documents and documents by words. In order to find the optimal K (number of clusters) for this exercise, we imported KElbowVisualizer from the Yellowbrick library. For reference, "Yellowbrick is an open source, pure Python project that extends the scikit-learn API with visual analysis and diagnostic tools" (Link: https://www.scikit-yb.org/en/latest/api/cluster/elbow.html).

Once we found the optimal K for each part of the exercise, we used K-means with the optimal K to cluster the data and then run the analysis. Our results are printed below along with the analysis of each approach and where it may be useful. Ultimately, while both approaches are insightful and shed light on word association in articles and suggestions for similar articles based on topics, we believe the stronger approach is to cluster based on documents by words.

For exercise 3, we implemented our own EM algorithm. The basic approach here is that we calculated the responsibilities (the weight of each point to each respective cluster) using our initial guesses for π_i, μ_i, and σ_i^2 in the E phase. In the M phase, we took the new π_i, μ_i, and σ_i^2 values and used them to recompute new responsibilities. We then repeated the E and M phases until our convergence criteria was met. In our case, we calculated the log likelihood at the end of each EM cycle and compared the current log likelihood to the previous one. Once this difference fell below 0.000000001, we could feel extremely confident that the data had converged and that changes were extremely close to 0. We produced plots to show the trajectories of our mean values during the runtime of the algorithm and also showed a distribution of the number of iterations it took to converge for a set of 50 different guesses for initial values.

We then used the concept of log likelihood to estimate initial guesses for our algorithm. To do this, we took the derivative of the log likelihood function with respect to μ and Σ and set those derivatives to 0. Although this will be discussed in more detail, it turns out that setting these derivatives equal to zero means we can estimate our μ and Σ values by just calculating the mean and variance for each feature for each cluster. When we do this, our algorithm converges in 7 iterations, which is much faster than the average we calculated for the 50 algorithm runs with initial guesses (average of 12.26). This gives us some insight into how choosing the initial guess might impact how long it takes to converge. However, regardless of how many iterations it took to converge, all cases converged to the same final values.

0.2 Problem 1: Eigenface for face recognition

In this assignment you will implement the Eigenface method for recognizing human faces. You will use face images from The Yale Face Database B, where there are 64 images under different lighting conditions per each of 10 distinct subjects, 640 face images in total. With your implementation, you will explore the power of the Singular Value Decomposition (SVD) in representing face images.

(a) Download The Face Dataset. After you unzip faces.zip, you will find a folder called images which contains all the training and test images; train.txt and test.txt specifies the training set and test (validation) set split respectively, each line gives an image path and the corresponding label.

(b) Load the training set into a matrix X: there are 540 training images in total, each has 50×50 pixels that need to be concatenated into a 2500-dimensional vector. So the size of X should be 540×2500, where each row is a flattened face image. Pick a face image from X and display that image in grayscale. Do the same thing for the test set. The size of matrix X_{test} for the test set should be 100×2500.

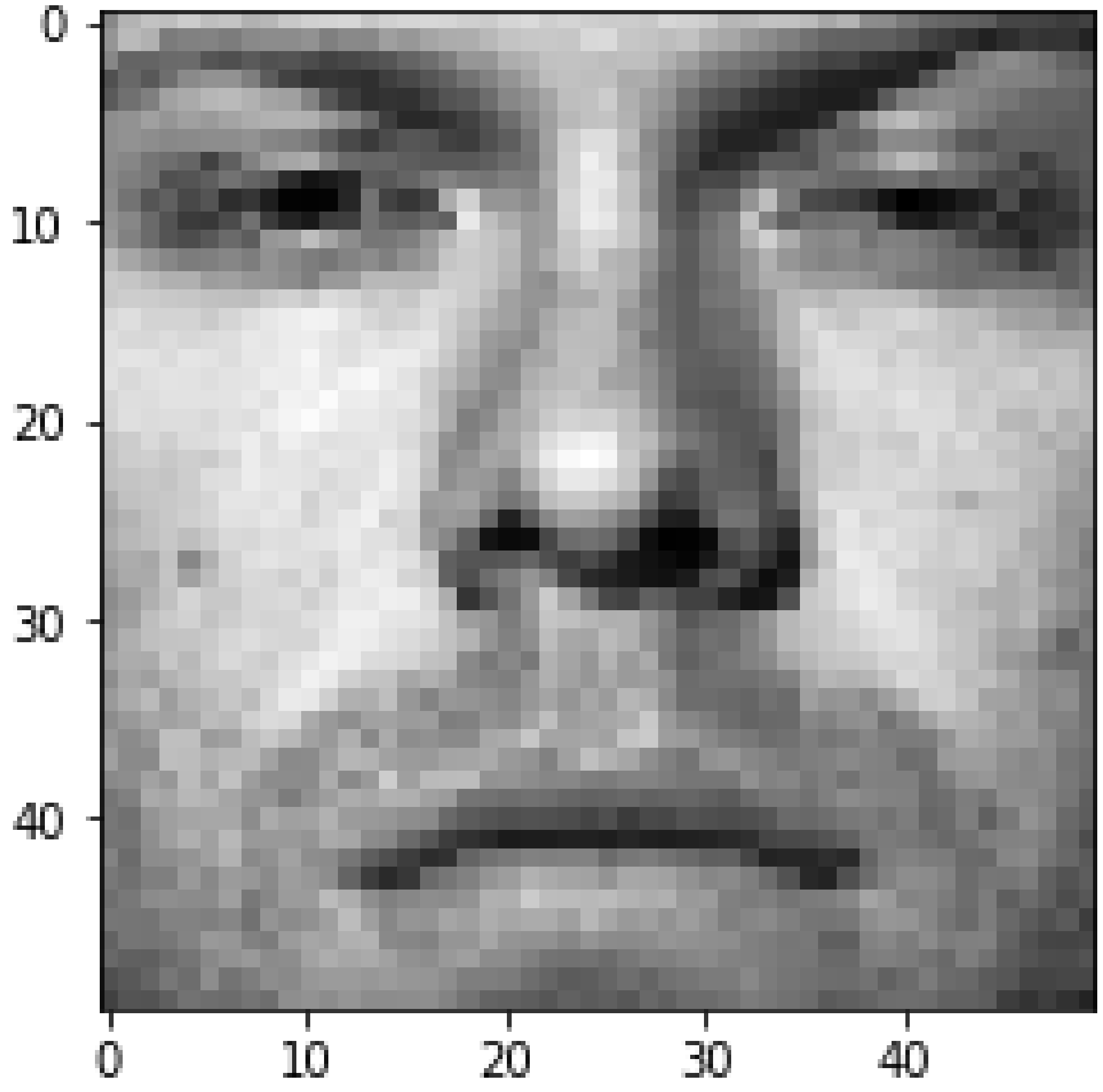


Figure 1: Face Image from X

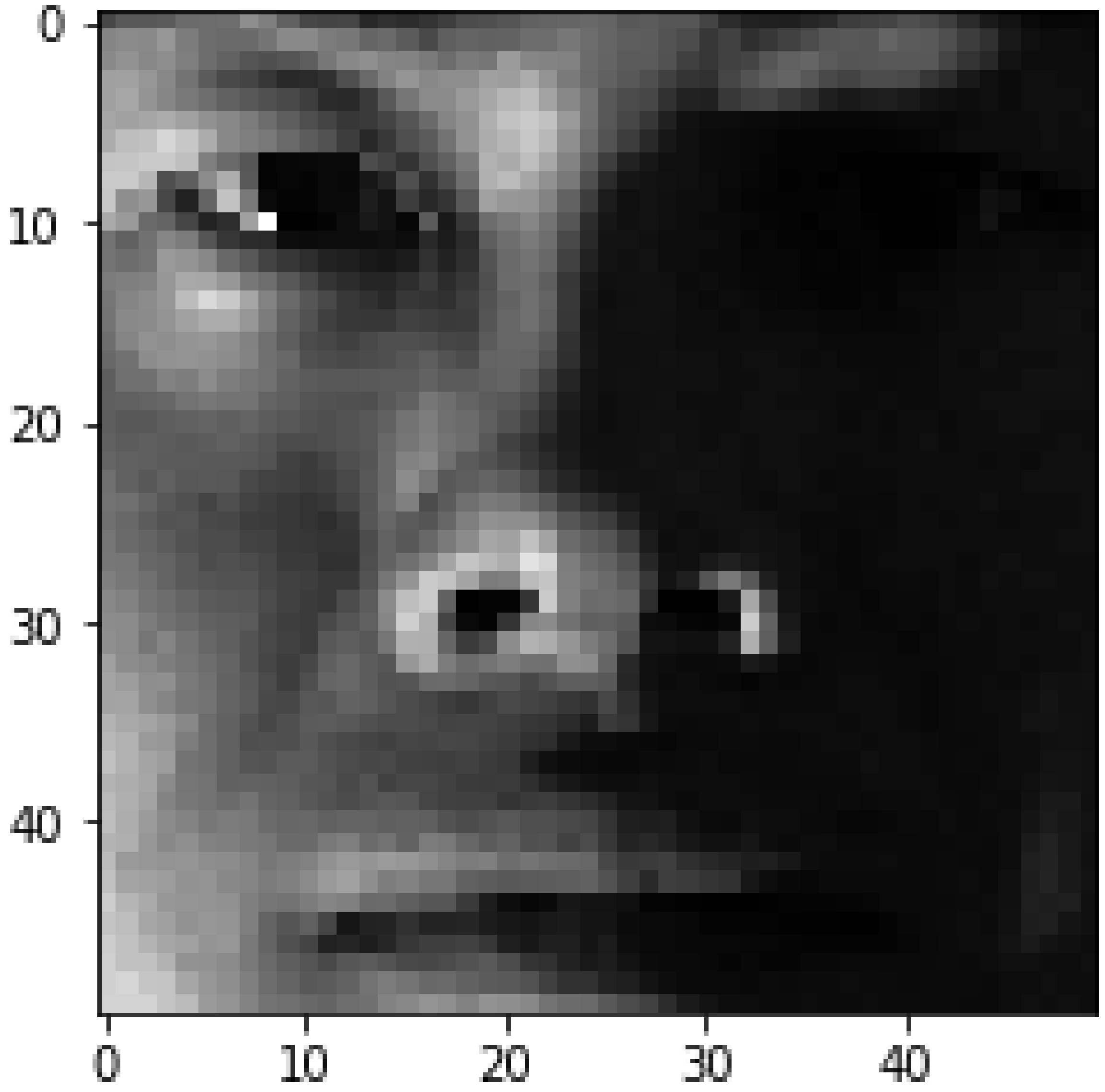


Figure 2: Face Image from X_{test}

(c) Average Face. Compute the average face µ from the whole training set by summing up every column in X then dividing by the number of faces. Display the average face as a grayscale image.

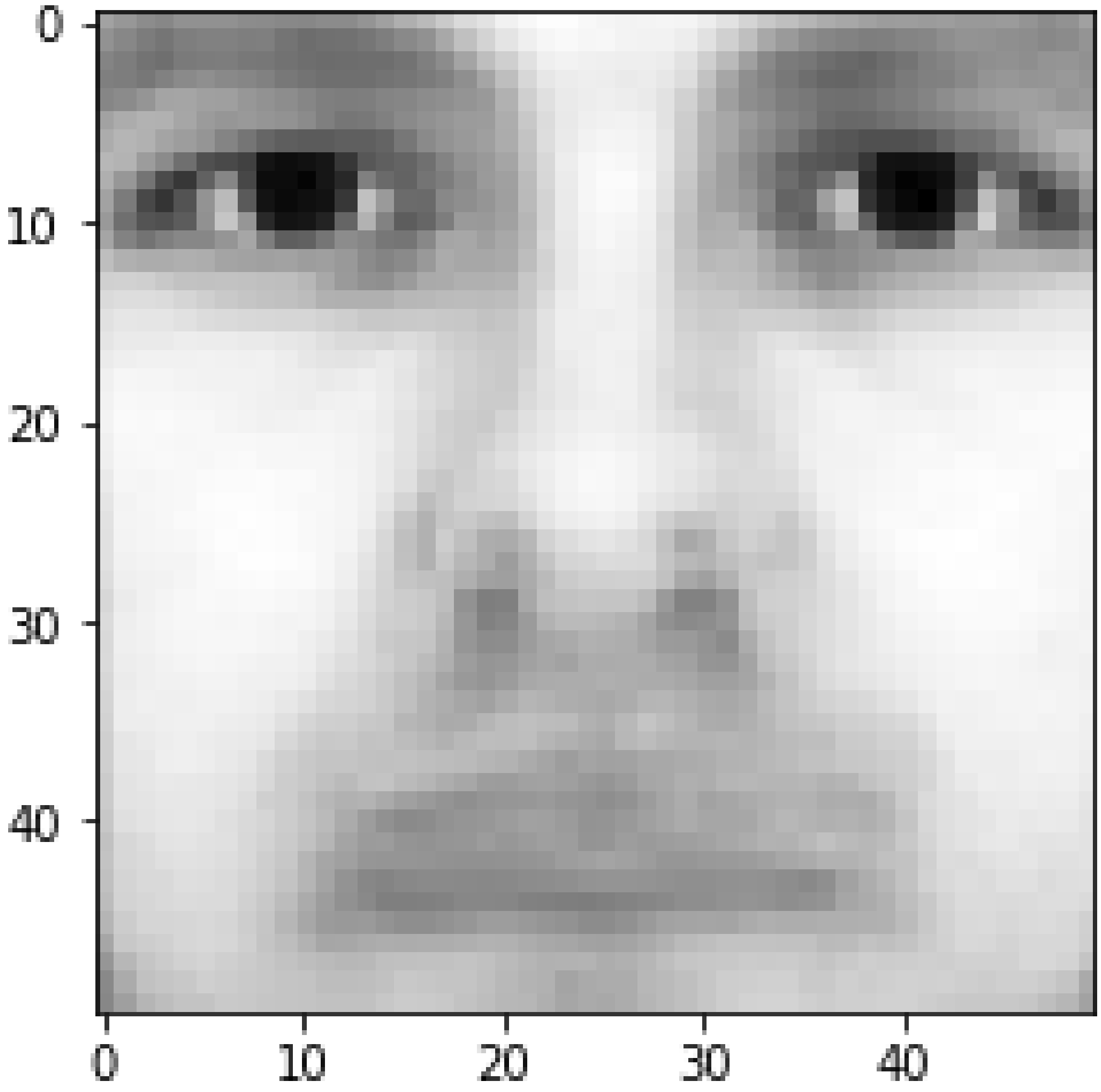


Figure 3: Average Face

(d) Mean Subtraction. Subtract average face µ from every column in X. That is, x_i:= x_i - µ, where x_i is the i-th column of X. Pick a face image after mean subtraction from the new X and display that image in grayscale. Do the same thing for the test set X_{test} using the precomputed average face µ in (c).

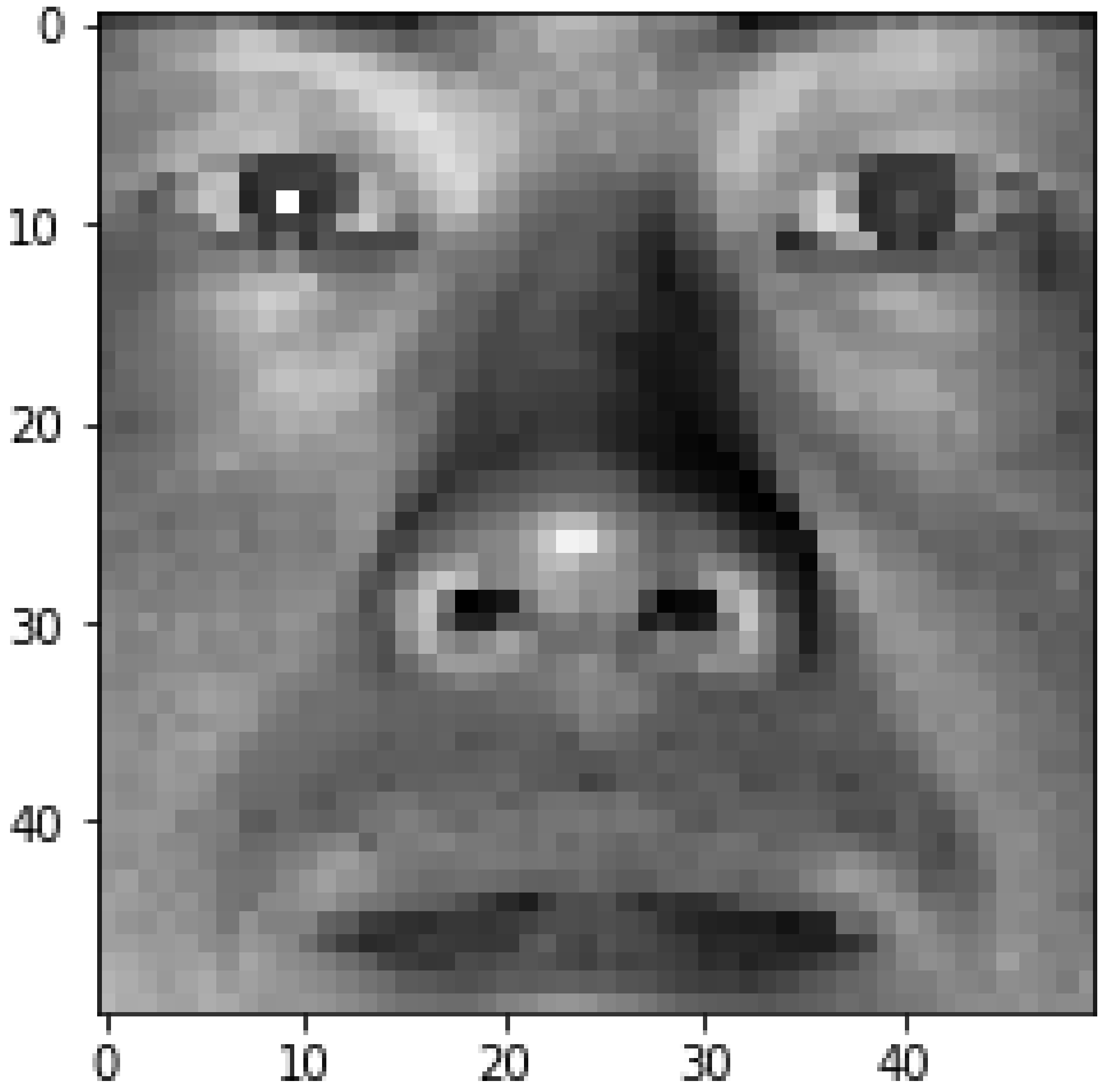


Figure 4: Training Face after Mean Subtraction

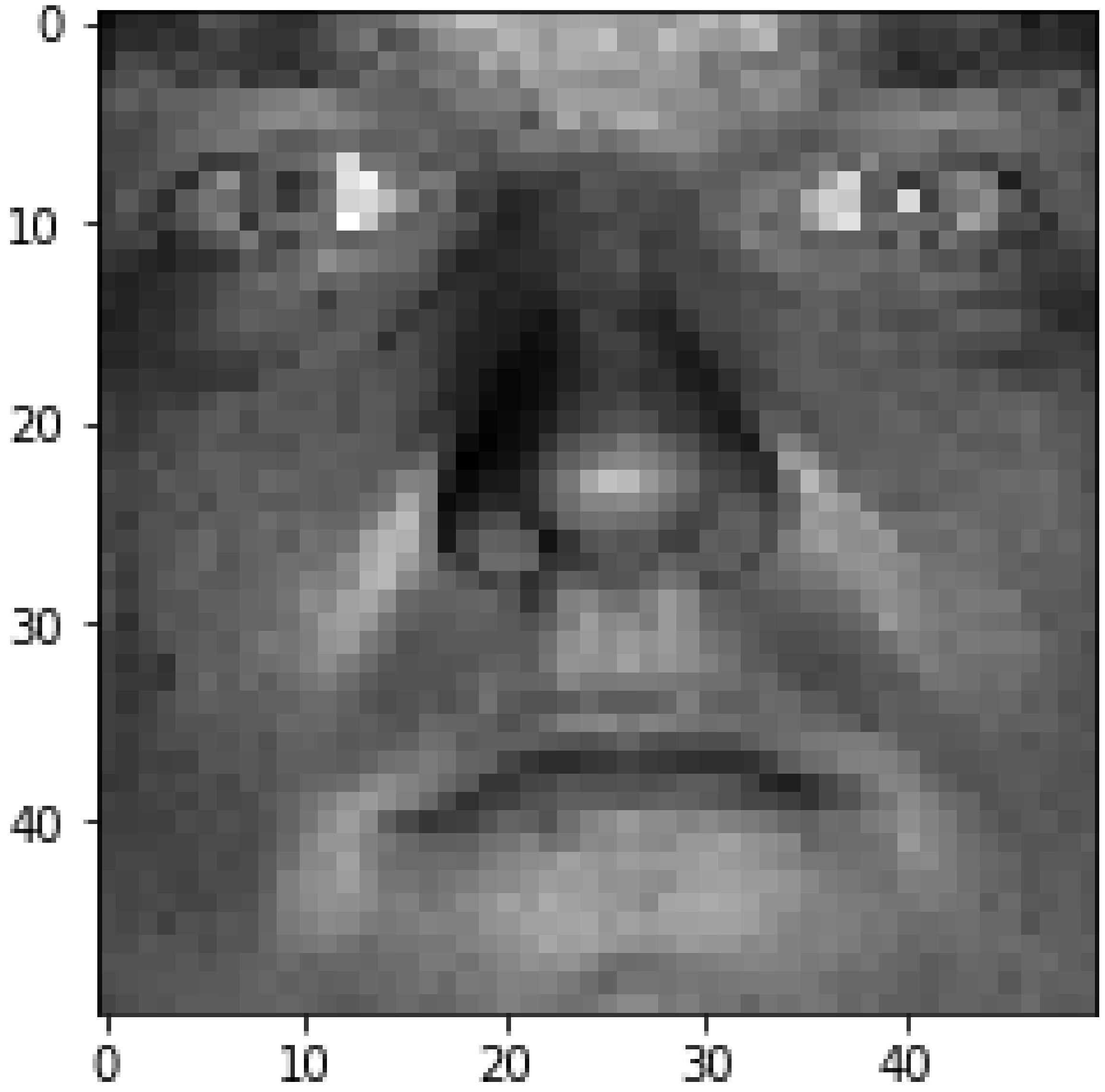


Figure 5: Test Face after Mean Subtraction

(e) Eigenface. Perform Singular Value Decomposition (SVD) on training set X (X = $U\Sigma V^T$) to get matrix V^T, where each row of V^T has the same dimension as the face image. We refer to v_i, the i-th row of V^T, as i-th eigenface. Display the first 10 eigenfaces as 10 images in grayscale.

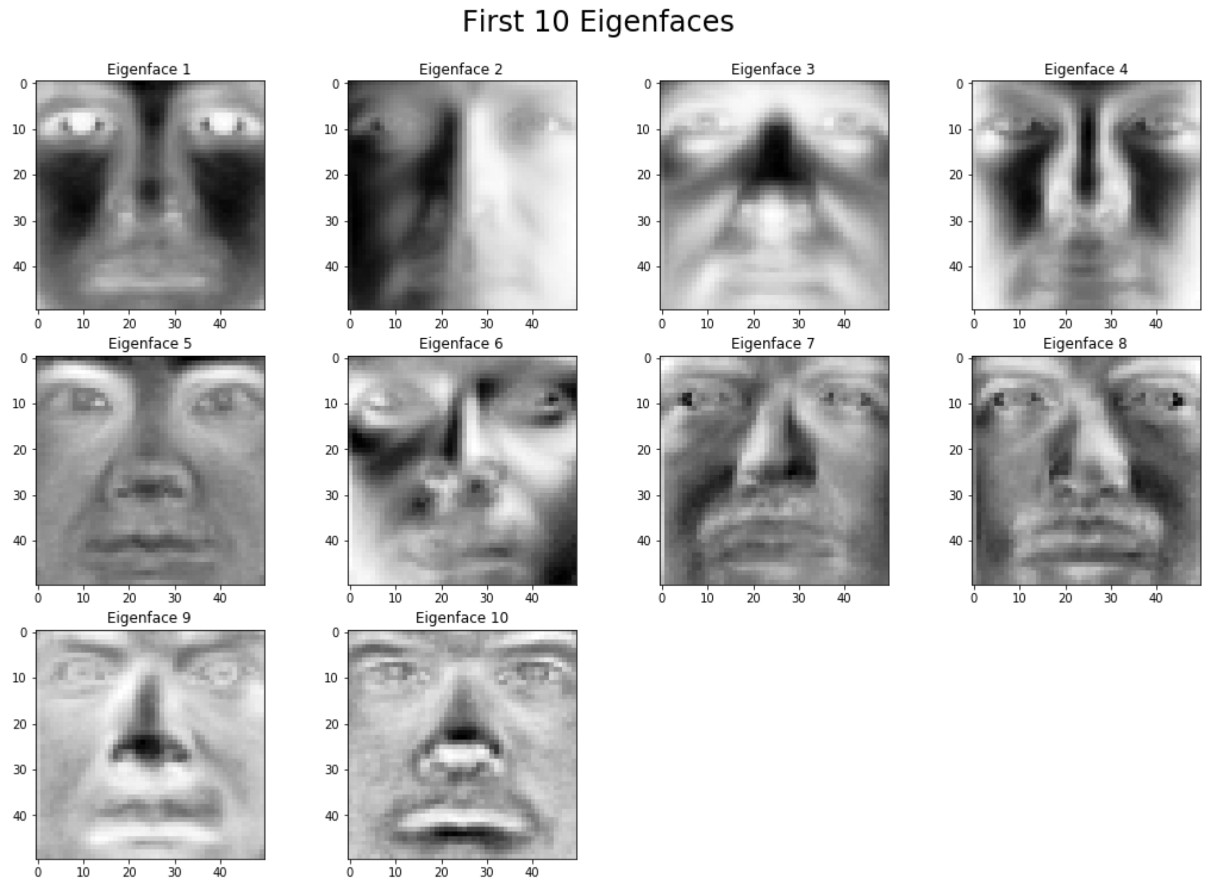


Figure 6: First 10 Eigenfaces

(f) Low-rank Approximation. Since Σ is a diagonal matrix with non-negative real numbers on the diagonal in non-ascending order, we can use the first r elements in Σ together with first r columns in U and first r rows in V^T to approximate X. That is, we can approximate X by $\hat{X}_r$ = U[:,: r] Σ[: r,: r] V^T[: r,:]. The matrix $\hat{X}_r$ is called rank-r approximation of X. Plot the rank-r approximation error $\left\| X - \hat{X}_r \right\|_F$ as a function of r when r = 1, 2,..., 200.

For this part, we found the rank approximation $\hat{X}_r$ = U[:,: r] Σ[: r,: r] V^T[: r,:] for each value of r between 1 and 200. We then took the frobenius norm of X - $\hat{X}_r$ for each r and plotted the distribution of this norm as r increases.

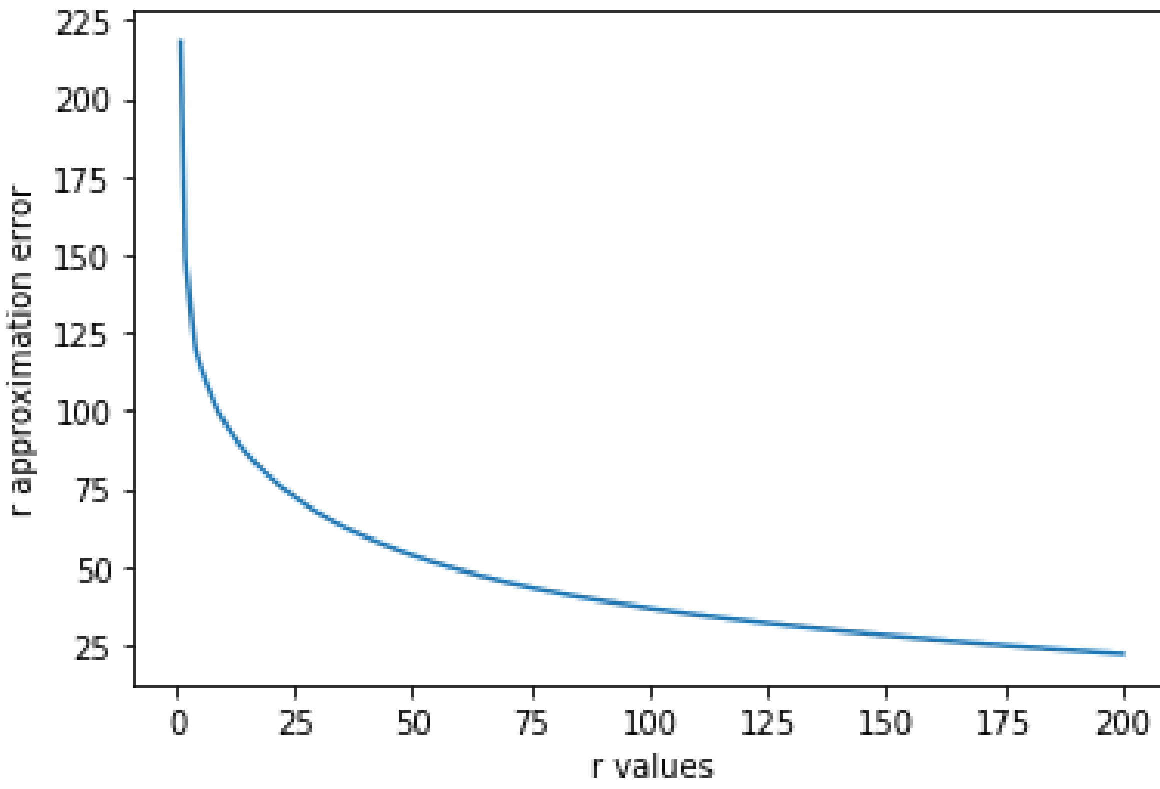


Figure 7: r approximation error for r = 1-200

As can be seen in the figure, our r approximation error decreases as we increase r, which makes sense as we are providing more features/information to our model. Specifically, the error seems to rapidly decrease between r = 0 and r = 10 and then the rate of error decrease begins to slow.

(g) Eigenface Feature. The top r eigenfaces V^T[: r,:] = $\{v_1, v_2,..., v_r\}^T$ span an r -dimensional linear subspace of the original image space called face space, whose origin is the average face μ, and whose axes are the eigenfaces $\{v_1, v_2,..., v_r\}$. Therefore, using the top r eigenfaces $\{v_1, v_2,..., v_r\}$, we can represent a 2500-dimensional face image z as an r -dimensional feature vector f: f = V^T[: r,:] z = $[v_1, v_2,..., v_r]^T$z. Write a function to generate r -dimensional feature matrix F and F_{test} for training images X and test images X_{test}, respectively (to get F, multiply X to the transpose of first r rows of V^T, F should have same number of rows as X and r columns; similarly for X_{test}).

> For this part, we define our function so that it receives the train and test data and an r value. We then perform SVD on the training and test sets. We calculate F by multiplying our training data by the V vector (just the first r rows of V^T) from our SVD transformation of our train data. Similarly, we calculate F-test by multiplying our test data by the V vector (just the first r rows of V^T) from our SVD transformation of our test data.

(h) Face Recognition. Extract training and test features for r = 10. Train a Logistic Regression model using F and test on F_{test}. Report the classification accuracy on the test set. Plot the classification accuracy on the test set as a function of r when r = 1, 2,..., 200. Use “one-vs-rest” logistic regression, where a classifier is trained for each possible output label. Each classifier is trained on faces with that label as positive data and all faces with other labels as negative data. sklearn calls this “ovr” mode.

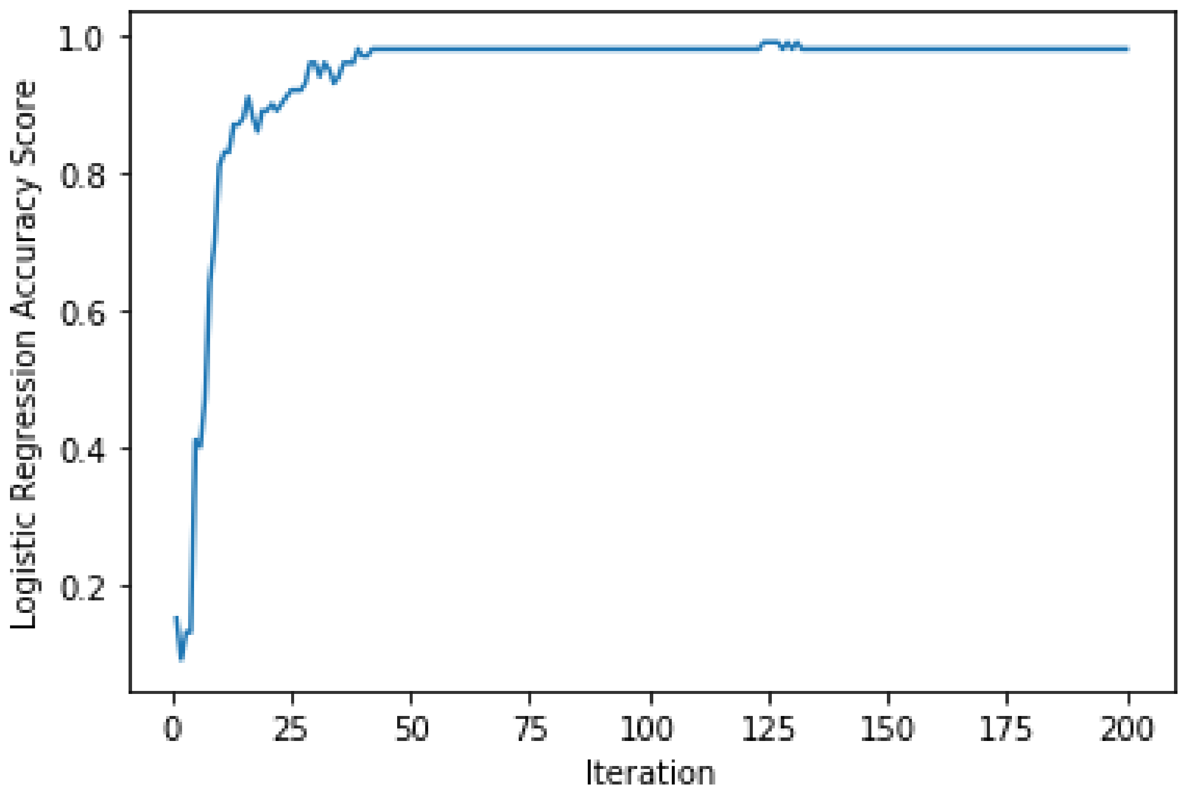


Figure 8: Logistic Regression Accuracy Score for r = 1 to 200

As our plot shows, when r is quite close to 0, our accuracy score for our logistic regression model is quite low/close to 0. However, at r = 10 we are already up to an accuracy score of 0.81. Additionally, the accuracy converges towards 0.98 as early as r = 50, which signifies that even just r = 50 features is enough to properly approximate our data.

0.3 Problem 2: Clustering for text analysis

In this problem, you will analyze all the articles from the journal Science in the year 2000. (Thanks to JSTOR for providing the data.) Many of the parameters of this analysis will be left for you to decide. For these files you will need to use science2k-vocab.npy and science2k-titles.npy, which are vectors of terms and titles respectively

(a) The file science2k-doc-word.npy contains a 1373×5476 matrix, where each row is an article in Science described by 5476 word features. The articles and words are in the same order as in the vocabulary and titles files above. You can read this file using numpy.load("science2k-doc-word.npy")

To obtain the features, we performed the following transformation. First, we computed perdocument smoothed word frequencies. Second, we took the log of those frequencies. Finally, we centered the per-document log frequencies to have zero mean.

Cluster the documents using k-means and various values of k (go up to at least k = 20). Select a value of k.

For that value, report the top 10 words of each cluster in order of the largest positive distance from the average value across all data. More specifically, if $\bar{x}$ is the 5476-vector of average values across documents and m_i is the ith mean, report the words associated with the top components in $m_i - \bar{x}$. Report the top ten documents that fall closest to each cluster center. You can find the titles in the science2k-titles.dat file. Comment on these results. What has the algorithm captured? How might such an algorithm be useful?

To begin, we focus on the the document by words case. In order to find the optimal K (number of clusters) for this exercise, we imported KElbowVisualizer from the Yellowbrick library. For reference, "Yellowbrick is an open source, pure Python project that extends the scikit-learn API with visual analysis and diagnostic tools." Through this tool we were able to graph and find an optimal K between 2 and 20 – in this case the optimal value is K = 10.

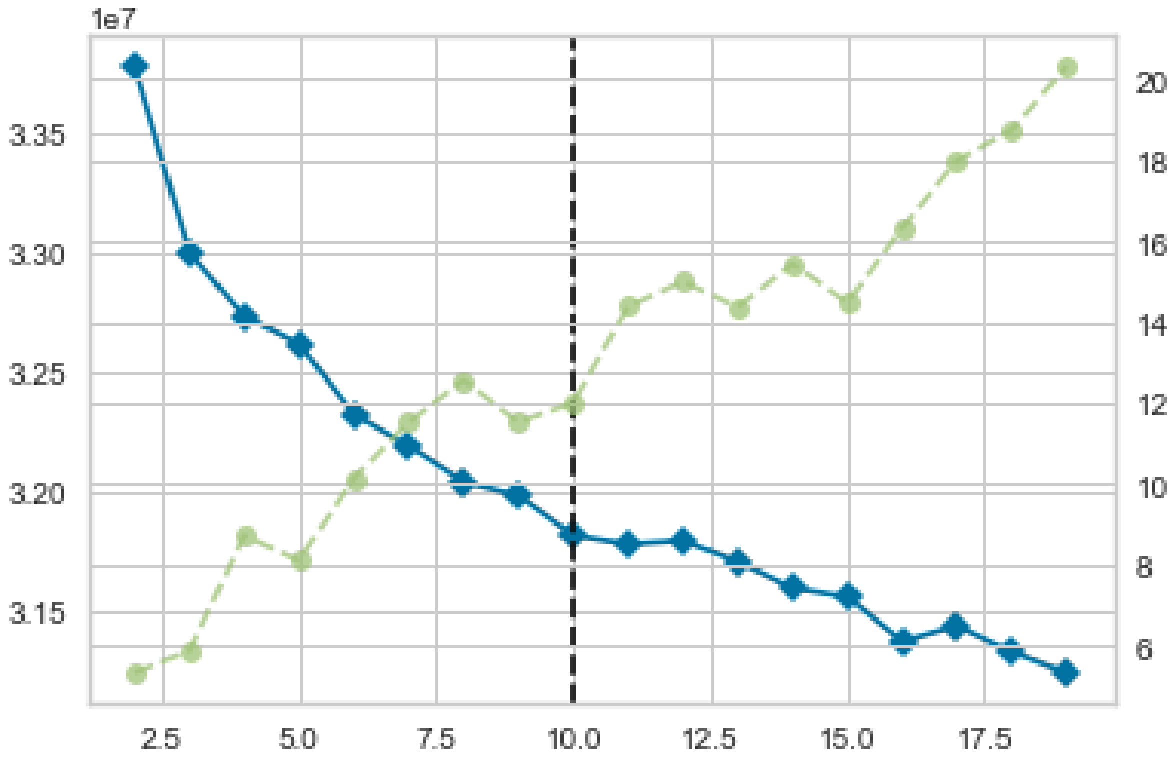
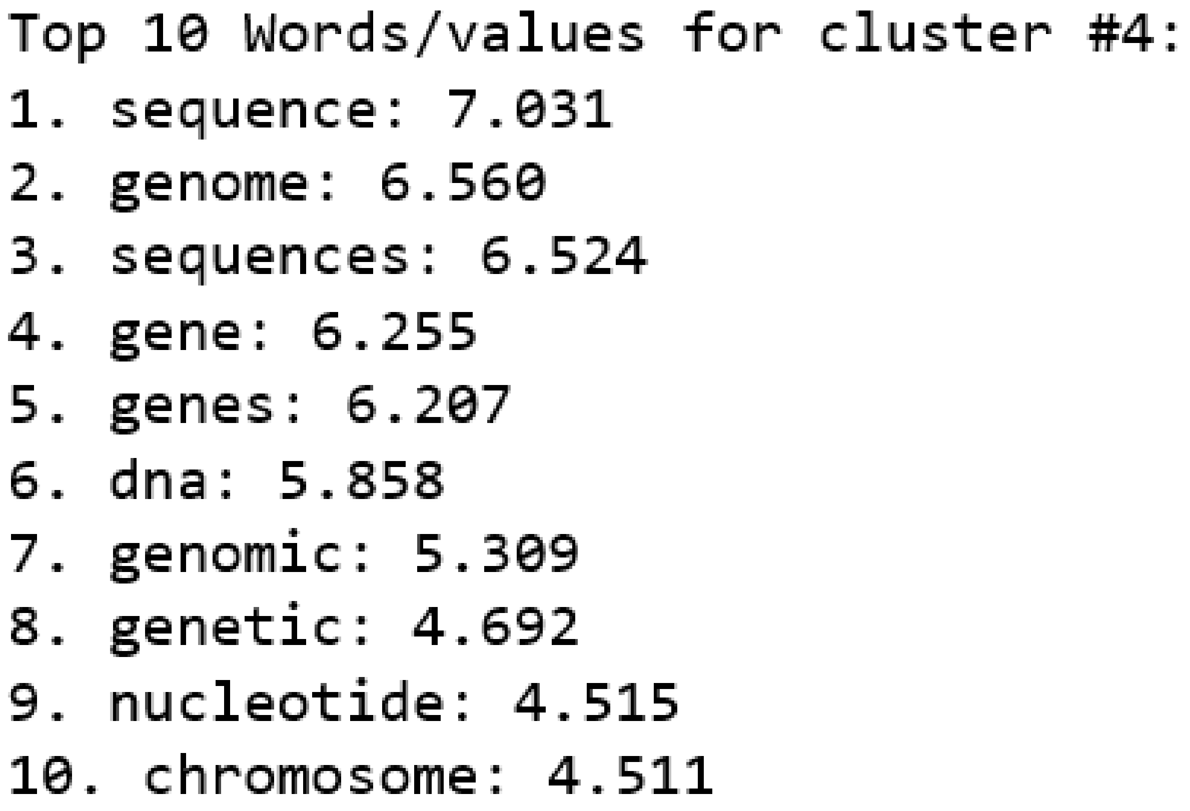


Figure 9: Optimal K clusters: Elbow Method Curve

For the first part of this exercise, we first perform K–means on our data. We then identified the words that appear most often in each cluster based on positive distance from the average value across all data. Below is a preview of the top 10 words for one cluster. Further supporting details for specific words per cluster can be found in the attached Jupyter notebook.



```
Top 10 Words/values for cluster #4:
1. sequence: 7.031
2. genome: 6.560
3. sequences: 6.524
4. gene: 6.255
5. genes: 6.207
6. dna: 5.858
7. genomic: 5.309
8. genetic: 4.692
9. nucleotide: 4.515
10. chromosome: 4.511
```

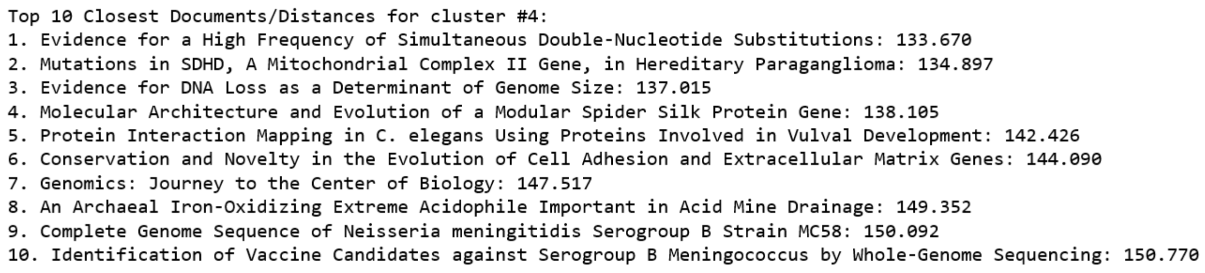
Figure 10: Top 10 Words for Cluster 4

The algorithm for the first part of (a) identifies the words that appear most frequently in the given cluster. Furthermore, the algorithm captures the following components, which make it useful:

1. Word association amongst clusters - one can see which words may complement each other when discussing certain topics (clusters). This could be used further in machine learning to predict genres/categories of articles based on top words and similarity to other feature vectors of articles

2. A lookup table based on keywords. For example, if I am curious to learn about genome, I may go to cluster 4 in our example and look at those given articles to learn more or see which words are frequently searched for along with it.

For the second portion of (a) we identify the 10 (or n if less than 10 documents in the cluster) documents closest to each cluster center. This exercise gives us a better sense for which topic/topics define the cluster.



```
Top 10 Closest Documents/Distances for cluster #4:
1. Evidence for a High Frequency of Simultaneous Double-Nucleotide Substitutions: 133.670
2. Mutations in SDHD, A Mitochondrial Complex II Gene, in Hereditary Paraganglioma: 134.897
3. Evidence for DNA Loss as a Determinant of Genome Size: 137.015
4. Molecular Architecture and Evolution of a Modular Spider Silk Protein Gene: 138.105
5. Protein Interaction Mapping in C. elegans Using Proteins Involved in Vulval Development: 142.426
6. Conservation and Novelty in the Evolution of Cell Adhesion and Extracellular Matrix Genes: 144.090
7. Genomics: Journey to the Center of Biology: 147.517
8. An Archaeal Iron-Oxidizing Extreme Acidophile Important in Acid Mine Drainage: 149.352
9. Complete Genome Sequence of Neisseria meningitidis Serogroup B Strain MC58: 150.092
10. Identification of Vaccine Candidates against Serogroup B Meningococcus by Whole-Genome Sequencing: 150.770
```

Figure 11: Top 10 Closest Documents for Cluster 4

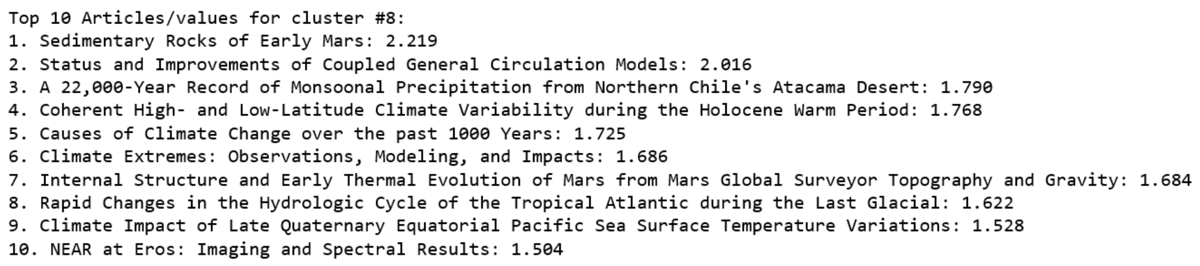
If we were curious to learn more about genomics, astronomy, or molecular biology, for example, we could look at these different documents, which would provide us with a cluster full of similar articles. This could be great when conducting research, as it could automatically provide suggestions for similar items that could further supplement the research.

Where as words may provide a more subtle view into the sentiment/tone of articles, which words may naturally complement each other when addressing certain topics, and what other words could be suggested to search for, clustering by documents provides us with more insight into the general theme of the cluster. Indeed, it is more focused on topic. In many ways, we get a bottom-up view with finding the most common words, while we get a nice top-down view when finding the documents closest to the cluster center.

(b) The file science2k-word-doc.txt is similar, but capture term-wise rather than documentwise features. That is, for each term, we count the frequency as the number of documents that term appears in rather than the other way around. This allows us to characterize individual terms.
This matrix is 5476×1373, where each row is a term in Science described by 1373 "document" features. These are transformed document frequencies (as above). Repeat the analysis above, but cluster terms instead of documents. The terms are listed in science2k-vocab.txt

Comment on these results. How might such an algorithm be useful? What is different about clustering terms from clustering documents?

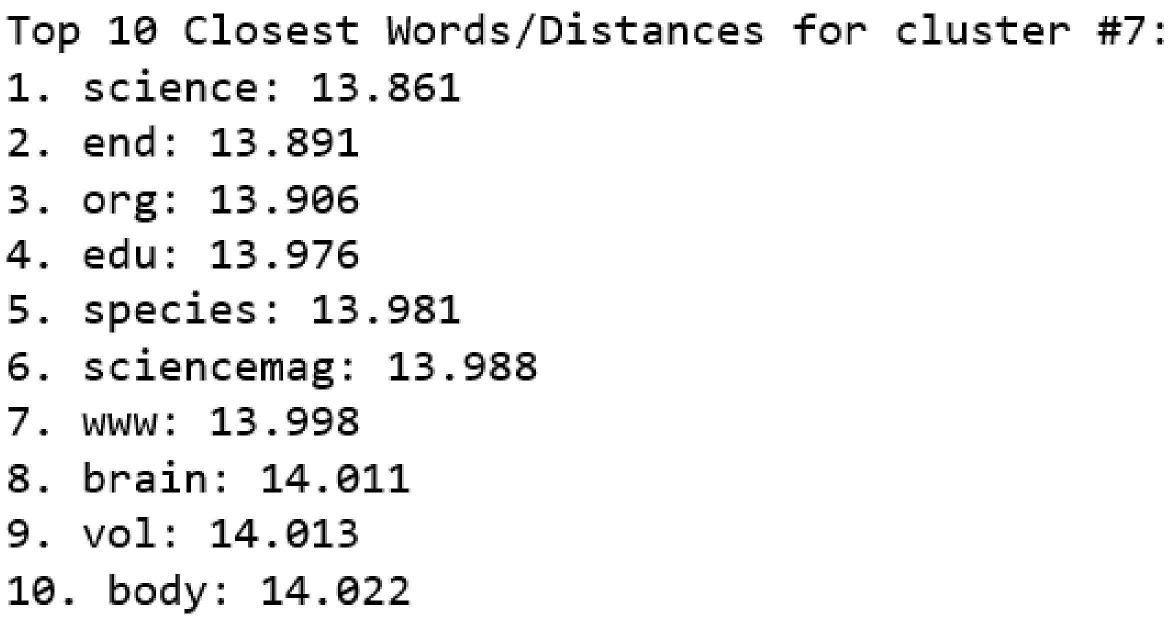
Similarly to part (a), we ran through a similar exercise, except the dimensions of the matrix data changed, along with the frequencies and such after smoothing. This is because we are focusing on the words by document case here. We first found the articles that appeared most often in relation to a given cluster in the dataset:



```
Top 10 Articles/values for cluster #8:
1. Sedimentary Rocks of Early Mars: 2.219
2. Status and Improvements of Coupled General Circulation Models: 2.016
3. A 22,000-Year Record of Monsoonal Precipitation from Northern Chile's Atacama Desert: 1.790
4. Coherent High- and Low-Latitude Climate Variability during the Holocene Warm Period: 1.768
5. Causes of Climate Change over the past 1000 Years: 1.725
6. Climate Extremes: Observations, Modeling, and Impacts: 1.686
7. Internal Structure and Early Thermal Evolution of Mars from Mars Global Surveyor Topography and Gravity: 1.684
8. Rapid Changes in the Hydrologic Cycle of the Tropical Atlantic during the Last Glacial: 1.622
9. Climate Impact of Late Quaternary Equatorial Pacific Sea Surface Temperature Variations: 1.528
10. NEAR at Eros: Imaging and Spectral Results: 1.504
```

Figure 12: Top 10 Documents for Cluster 8

We then also found the words closest to the cluster center and observed which words gravitated to certain cluster centers. If we look at cluster 4 for example, we see the words cells, gene, dna, mice, and proteins all in one cluster. We can suspect that this cluster's focus is driven by articles focusing on medical research for example. Further, we noticed that for cluster 7, for example, that the distances were much less for that cluster, leading us to believe that, generally speaking, that cluster was more compact than others.



```
Top 10 Closest Words/Distances for cluster #7:
1. science: 13.861
2. end: 13.891
3. org: 13.906
4. edu: 13.976
5. species: 13.981
6. sciencemag: 13.988
7. www: 13.998
8. brain: 14.011
9. vol: 14.013
10. body: 14.022
```

Figure 13: Top 10 closest words for Cluster 7

When completing this exercise and looking at the results, we observed that we may know which words are closest to the cluster center, but in this case we don't necessarily have a sense for the context of that cluster center. However, this algorithm could be useful when wanting to observe how words are associated with specific fields more than others.

In summary, from a high-level perspective, the two approaches in a and b differ, in (a) the approach is driven by the words, which influence which documents are clustered together, where as in (b) the approach is driven by articles, which dictate which words are clustered together. Generally speaking, we believe we would be more interested in using the approach in (a) to know which words showed up most frequently in a cluster and which articles clustered together in order to get the context for what a given cluster's theme is and not to guess what the theme is driven by the words that are clustered together. Ultimately, we think a cluster of documents gives us more context than a cluster of words.

0.4 Problem 3: EM algorithm/implementation

(a) The parameters of Gaussian Mixture Model (GMM) can be estimated via the EM algorithm. Show that the alternating algorithm for k-means (in Lec. 11) is a special case of the EM algorithm and show the corresponding objective functions for E-step and M-step.

Below we briefly explain what both k-means and GMM are and how they work.

k-means: k-means is a methodology to identify k clusters within a dataset, such that the clusters best present data points that are similar to each other to be in the same cluster. The high-level methodology of the alternating algorithm is that in step 1, centroids (center points) for each cluster are chosen randomly and each point in the dataset is then compared to each centroid to see which it is closest to (given a distance measurement) and assigned to that cluster. This methodology is known as hard clustering, as the points are assigned to a given cluster. After this step has completed, we will proceed to step 2, where we recompute the centroids to be the mean of the given cluster they represent - the objective here is to minimize the variance in the cluster as much as possible - to make it as compact as possible. This process will repeat until convergence is reached when the variance amongst each cluster does not improve.

GMM: GMM is a methodology to identify k clusters within a dataset, such that the clusters best present data points that are similar to each other to be in the same cluster. The major difference between GMM and k-means is that GMM is referred to as soft clustering, whereas k-means is hard. Furthermore, soft clustering means that we look for the mixture/responsibility of each data point to the given clusters in the dataset. That is to say, we calculate how likely the data point is to be in cluster a vs. b vs. c and it has a responsibility for each, so it does not belong to one cluster. Rather, it belongs to a mixture of clusters each with a different responsibility. The high-level methodology of the EM algorithm is that we first pick random or arbitrary mean, standard deviation, and pi. Then in the E step we calculate the responsibilities for each data point to determine the likelihood that it is part of any given cluster. After this step has completed, we will proceed to the M step, where we recompute the means, standard deviations, and pis - the objective here is to maximize the log likelihood for each point for all clusters This process will repeat until convergence is reached when the log likelihood does not increase further.

Ultimately, we can see several notable items from above. Both algorithms aim to cluster data, and both alternate between two steps with similar objectives - Step 1 and the E step are where the distances from centroids and cluster responsibilities are calculated and step 2 and M step are where we move around the centroids and cluster responsibilities to **minimize** variance in the cluster and **maximize** log likelihood, respectively. Thus, we see that the objective functions are very much the same, though with slightly different approaches to fit the hard vs. soft clustering requirements and to **minimize** vs. **maximize**.

(b) Download the Old Faithful Geyser Dataset. The data file contains 272 observations of (eruption time, waiting time). Treat each entry as a 2 dimensional feature vector. Parse and plot all data points on 2-D plane.

After parsing and plotting our 2-dimensional data (each point has an eruption value and a waiting value), we notice that the data naturally clusters into two visibly distinct sections (though there is certainly overlap).

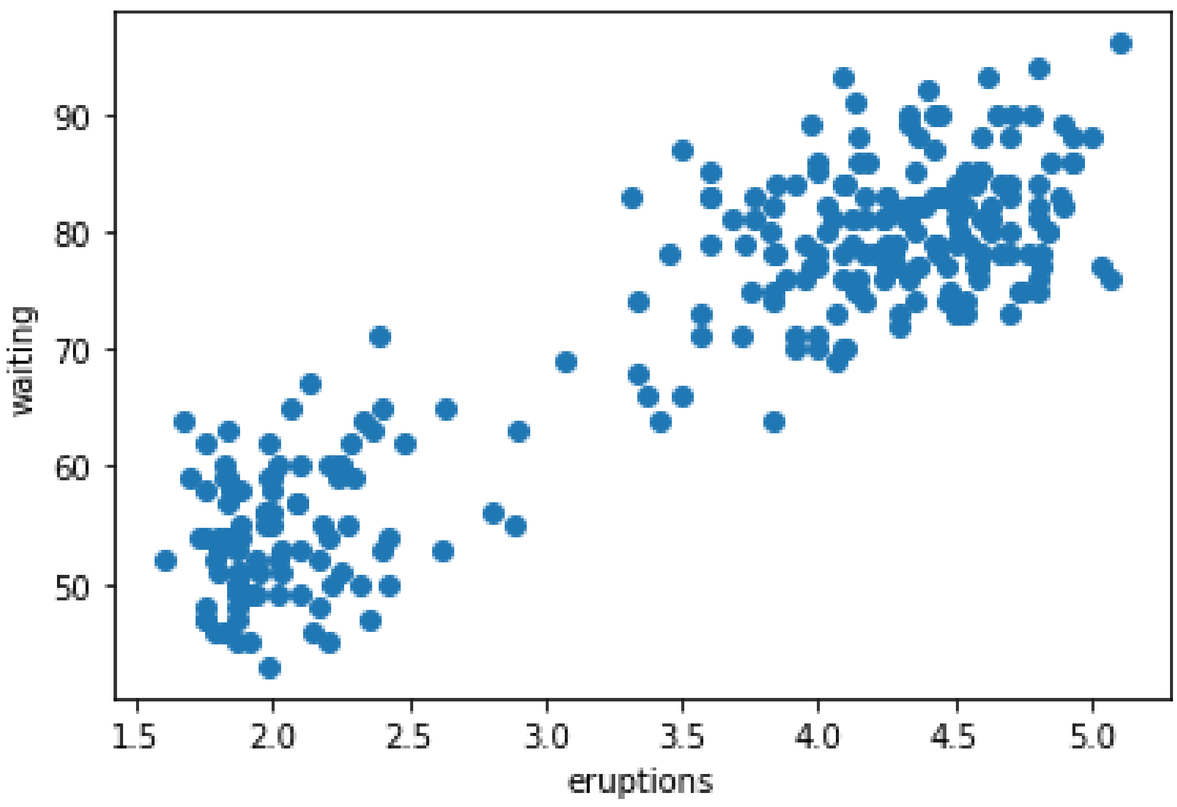


Figure 14: Eruption and Waiting Data

(c) Implement a bimodal GMM model to fit all data points using EM algorithm. Explain the reasoning behind your termination criteria. For this problem, we assume the covariance matrix is spherical (i.e., it has the form of σ^2I for scalar σ and you can randomly initialize Gaussian parameters. For evaluation purposes, please submit the following figures:

- Plot the trajectories of two mean vectors in 2 dimensions (i.e., coordinates vs. iteration).
- Run your program for 50 times with different initial parameter guesses. Show the distribution of the total number of iterations needed for algorithm to converge

In order to implement our EM algorithm, we made use of Nathan's class notes and several additional sources listed below:

- http://cs229.stanford.edu/section/gaussians.pdf?fbclid=IwAR2HuwS7 L20Gkszga95Bd2CxQpC_eH6cQ8BaQMxz9XIm1-V-BunLHsYqR4A

- https://www.youtube.com/watch?v=qMTuMa86NzU (Presentation by Professor Alexander Ihler from UC Irvine)

Our algorithm follows in the following steps:

1. An initial guess for the following paramaters are made: π_i, μ_i, and σ_i^2, where i is either cluster 1 or 2. Since our X data has two features, our μ_i and σ_i^2 values will also be vectors with two values, one for each feature.
2. The E step of the algorithm is run. The π_i, μ_i, and σ_i^2 values are used to calculate the responsibility values for each cluster, r_1 and r_2.
3. The M step of the algorithm is run. The r_1 and r_2 values are used to calculate new values for π_i, μ_i, and σ_i^2.
4. Steps 2 and 3 continue repeating until a convergence criteria is met. Our criteria will be discussed further down in the response.

For the E phase, we start by calculating our N (probability density) for a gaussian model where X has two features. We draw upon the equation from the stanford article:

$$N_i = \frac{1}{\sqrt{2\pi\sigma_1^2}} \exp(-\frac{1}{2\sigma_1^2}(x_1 - \mu_1)^2) \cdot \frac{1}{\sqrt{2\pi\sigma_2^2}} \exp(-\frac{1}{2\sigma_2^2}(x_2 - \mu_2)^2).$$

For this equation, we have an N value for each cluster (N_1 and N_2). To calculate the r values for the E phase, we use the following equation with our N and π values:

$$r_1 = \frac{\pi_1 N_1}{\pi_1 N_1 + \pi_2 N_2},\ r_2 = \frac{\pi_2 N_2}{\pi_1 N_1 + \pi_2 N_2}$$

Once we have our responsibility arrays, we can transition to the M phase. In the M phase, we use the responsibilities for each cluster and the X data to calculate new π_i, μ_i, and σ_i^2 values. We make use of the following equations (from Nathan's lecture notes and our Youtube video source):

$m_i = \sum r_i$ (cluster i total responsibility – sum all values in resp. array)

$\pi_i = \frac{m_i}{m}$ (where m is sum of m_1 and m_2)

$\mu_i = \frac{1}{mi} \sum r_i X$ (since X has 2 features, μ_i will be a vector of length 2)

$\sigma_i^2 = \frac{1}{mi} \sum r_i (X - \mu_i)^2$ (since X has 2 features, σ_i^2 will be a vector of length 2)

With these calculations, we derive new π_i, μ_i, and σ_i^2 that we can plug back into our E phase and can continue repeating this cycle between E and M phases. Therefore, the next step is to figure out at what point our initial guesses have converged to final values, which will indicate to us that our E-M cycle should stop running. To do this we make use of the log likelihood calculation found in the Youtube video:

log-likelihood $= \sum \log_{10}(\pi_1 N_1 + \pi_2 N_2)$,

where we sum this calculation for all values in the N arrays. In our approach, we define convergence as when:

log-likelihood$_{current}$ - log-likelihood$_{previous}$ < 0.000000001

Our choice of 0.000000001 is just an arbitrarily low value. This is because if the change between two iterations is less than such a small value, we can be confident that the values have converged and that the change between the two steps is essentially 0.

To initially test our algorithm, we provided the following initial guesses:

- μ_1 = [0.0, 30]
- μ_2 = [2.0, 10]
- σ_1^2 = [1.0, 10]
- σ_2^2 = [0.5, 60]
- π_1 = 0.7

- $\pi_2 = 0.3$

Upon convergence, our program then provided the following outputs/results:

- count = 15
- $\mu_1 = [2.038, 54.493]$
- $\mu_2 = [4.291, 79.986]$
- $\sigma_1^2 = [0.070, 33.756]$
- $\sigma_2^2 = [0.168, 35.773]$
- $\pi_1 = 0.357$
- $\pi_2 = 0.643$

As our results indicate, the algorithm takes 15 iterations to converge. Additionally, it provides a lot of insight into our bi variate distributions, particularly the means and variances. For this particular usage of our EM algorithm, we can also plot the trajectories of the means with each iteration. In our plot, we show how the mean for each feature in each cluster shifts with each iteration until convergence. We plot the eruption feature for both clusters 1 and 2 in one plot and the waiting feature for both clusters 1 and 2 in the other plot:

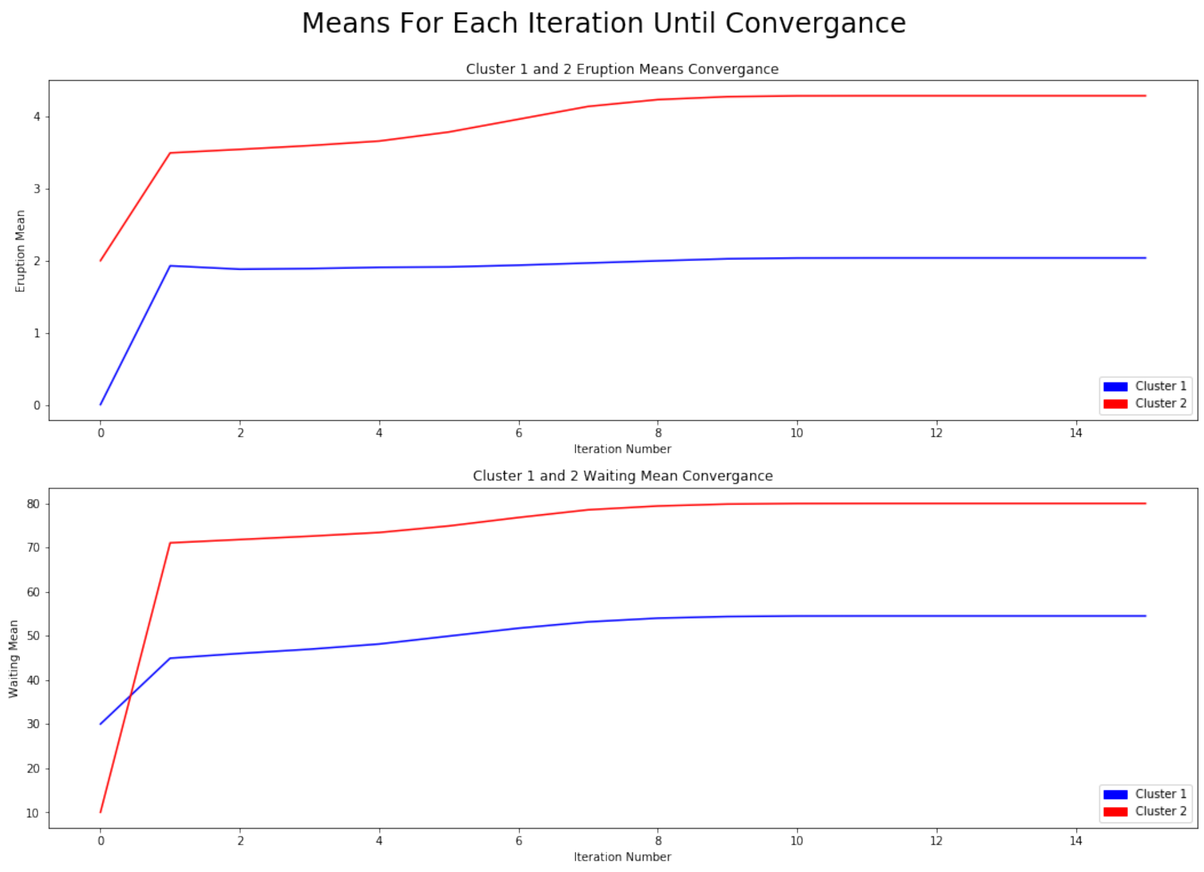


Figure 15: Mean Trajectories With Each Iteration

As can be seen, all of the means take a few iterations to reach a certain plateau, at which point they begin fluctuating less and less until the iteration-to-iteration fluctuation is less than the convergence threshold we set earlier.

To get a better idea of how our algorithm broadly performs, we run the same algorithm 50 times so that each time through generates new values for the initial guess set. By doing this, we can better capture how many iterations it takes to converge on average among 50 random initial guess sets, which can be seen in the resulting plot:

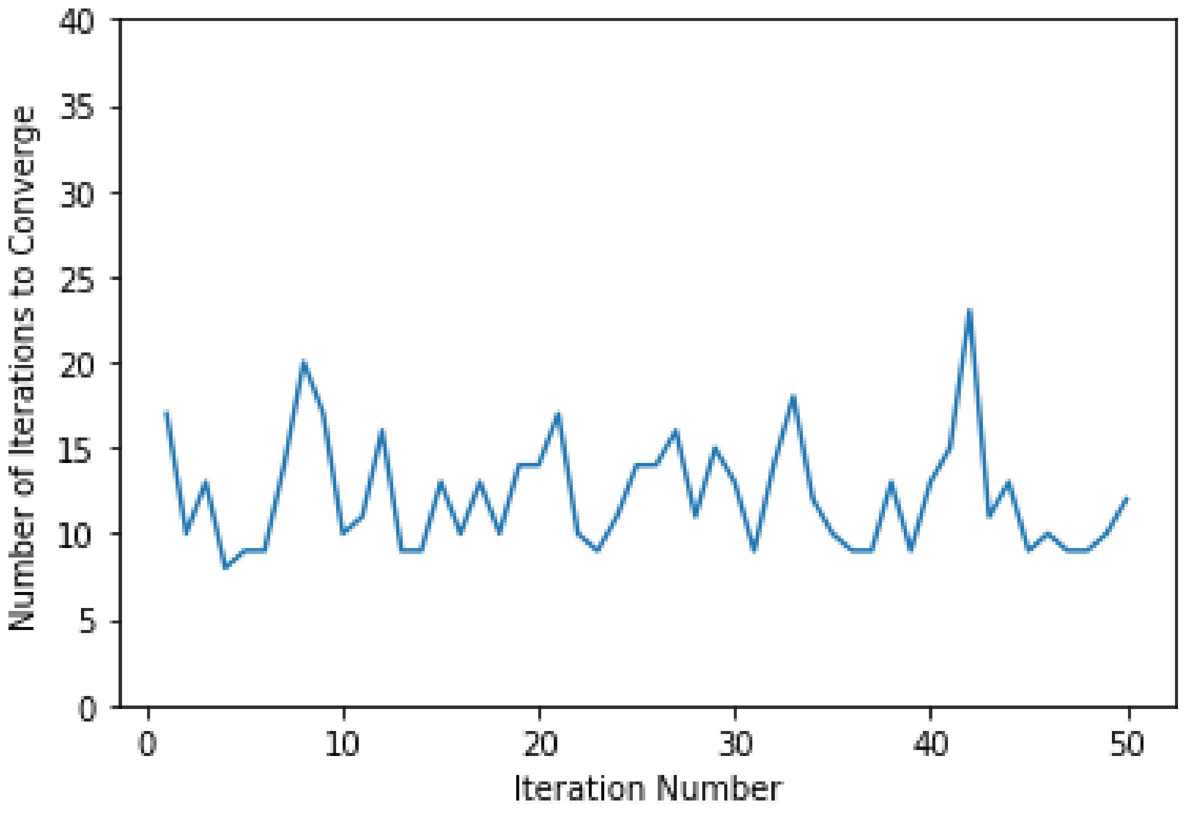


Figure 16: Number of Iterations to Converge for 50 Different EM Algorith Runs

From this plot, we can gather several insights about our algorithm. First, the values for number of iterations until convergence tends to range between approximately 8 to 25 for our 50 runs. Additionally, it only seems to hit a value of 8 once. We can also take the average of the number of iterations, which turns out to be 12.26 iterations until convergence (for the 50 runs).

(d) Repeat the task in (c) but with the initial guesses of the parameters generated from the following process:

- Run a k-means algorithm over all the data points with K = 2 and label each point with one of the two clusters.
- Estimate the first guess of the mean and covariance matrices using maximum likelihood over the labeled data points.

Compare the algorithm performances of (c) and (d).

For this part, we will still want to run our EM algorithm with an initial guess. However, we will use K-Means and Max-Likelihood to establish what our initial guess values should be. Using K-Means, we label our data so that it is split into two clusters. In this case, we define our π_i values as:

$$\pi_1 = \frac{\text{\# of points in cluster 1}}{\text{\# of points total}},\ \pi_2 = \frac{\text{\# of points in cluster 2}}{\text{\# of points total}}$$

For estimating our guess values for μ_i and σ_i^2, we made use of the max-likelihood method, using the following source as reference:

- https://www.cs.cmu.edu/ epxing/Class/10701-08s/recitation/gaussian.pdf

As is described by the source, we take the log likelihood function and take its derivative with respect to μ and then separately take its derivative with respect to Σ^{-1}. If we set both of these derivatives to 0, we are finding the points where μ and Σ are maximized, as a derivative of 0 represents a potential maxima. Using this concept and following the derivation in the CMU source, our equations derived from log likelihood simplify to:

$$\mu = \frac{1}{N}\sum_{n=1}^{N} X_n$$

$$\Sigma^{-1} = \frac{1}{N}\sum_{n=1}^{N}(X_n - \mu)^2$$

Since our X data has two features and since we have two clusters, we end up calculating a mean and variance for each cluster for each feature. As our log likelihood derivation shows, our ideal mean and variance values are equal to simply taking the mean and variance of the data for the specific cluster and feature we are interested in. Using this concept, we calculated our estimate for the μ_i and σ_i^2 values of our initial guess. The results of running our EM algorithm with the initial guess calculated through this K-Means and Maximum Likelihood method are below:

- count = 7
- $\mu_1 = [2.038, 54.493]$
- $\mu_2 = [4.291, 79.986]$
- $\sigma_1^2 = [0.070, 33.756]$
- $\sigma_2^2 = [0.168, 35.773]$
- $\pi_1 = 0.357$
- $\pi_2 = 0.643$

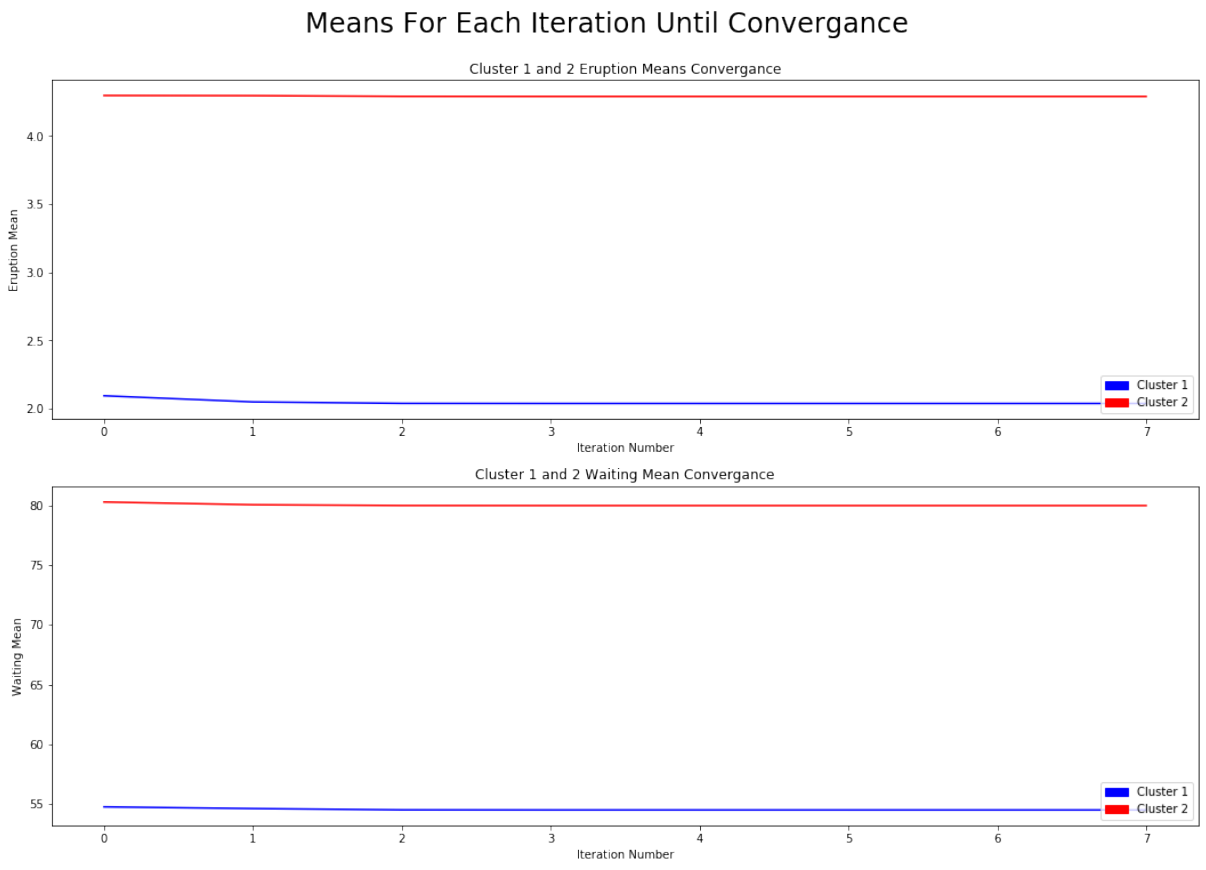


Figure 17: Mean Trajectories With Each Iteration

The results are very telling. First, the same values for μ_1, μ_2, σ_1^2, σ_2^2, π_1, π_2 are calculated upon convergence, which further confirms that our algorithm arrives at the same point regardless of the initial guess. Even more telling is that it only required 7 iterations to converge, which is less than any of the 50 runs we performed in part c. The average for the 50 runs in part c was 12.26, so the number of iterations is significantly lower than the average in this case as well. Looking at our mean trajectories as well also highlights that there is much less fluctuation than in part c. Since the values are estimated so that they start very close to the value they will ultimately converge to, the fluctuations are much smaller and the number of iterations needed to converge is much smaller.

0.5 Problem 4: Written Exercises

1. SVD of Rank Deficient Matrix. Consider matrix M. It has rank 2, as you can see by observing that three times the first column minus the other two columns is 0.

$$M = \begin{bmatrix} 1 & 0 & 3 \\ 3 & 7 & 2 \\ 2 & -2 & 8 \\ 0 & -1 & 1 \\ 5 & 8 & 7 \end{bmatrix}$$

(a) Compute the matrices $M^T M$ and MM^T.

First, we take the transpose of M:

$$M^T = \begin{bmatrix} 1 & 3 & 2 & 0 & 5 \\ 0 & 7 & -2 & -1 & 8 \\ 3 & 2 & 8 & 1 & 7 \end{bmatrix}$$

Next, we use the principle of matrix multiplication to calculate $M^T M$. Our individual calculations for each $(M^T M)_{ij}$ are shown as well:

$$M^T M = \begin{bmatrix} 1+9+4+0+25 & 0+21-4+0+40 & 3+6+16+0+35 \\ 0+21-4+0+40 & 0+49+4+1+64 & 0+14-16-1+56 \\ 3+6+16+0+35 & 0+14-16-1+56 & 9+4+64+1+49 \end{bmatrix}$$

$$M^T M = \begin{bmatrix} 39 & 57 & 60 \\ 57 & 118 & 53 \\ 60 & 53 & 127 \end{bmatrix}$$

Next, we again use the principle of matrix multiplication to calculate MM^T. Our individual calculations for each $(MM^T)_{ij}$ are shown as well:

$$MM^T = \begin{bmatrix} 1+0+9 & 3+0+6 & 2+0+24 & 0+0+3 & 5+0+21 \\ 3+0+6 & 9+49+4 & 6-14+16 & 0-7+2 & 15+56+14 \\ 2+0+24 & 6-14+16 & 4+4+64 & 0+2+8 & 10-16+56 \\ 0+0+3 & 0-7+2 & 0+2+8 & 0+1+1 & 0-8+7 \\ 5+0+21 & 15+56+14 & 10-16+56 & 0-8+7 & 25+64+49 \end{bmatrix}$$

$$MM^T = \begin{bmatrix} 10 & 9 & 26 & 3 & 26 \\ 9 & 62 & 8 & -5 & 85 \\ 26 & 8 & 72 & 10 & 50 \\ 3 & -5 & 10 & 2 & -1 \\ 26 & 85 & 50 & -1 & 138 \end{bmatrix}$$

(b) Find the eigenvalues for your matrices of part (a).

In order to calculate the eigenvalues (and eigenvectors), we use the following python lines of code:

eigen_vals1, eigen_vecs1 = np.linalg.eig(M_MT)
eigen_vals2, eigen_vecs2 = np.linalg.eig(MT_M)

The results are the following:

$$\lambda_{MM^T} = \begin{bmatrix} 214.67 & 0 & 69.33 & 0 & 0 \end{bmatrix}$$

$$\lambda_{M^TM} = \begin{bmatrix} 214.67 & 0 & 69.33 \end{bmatrix}$$

(c) Find the eigenvectors for the matrices of part (a).

In order to calculate the eigenvectors (and eigenvalues), we use the following python lines of code:

eigen_vals1, eigen_vecs1 = np.linalg.eig(M_MT)
eigen_vals2, eigen_vecs2 = np.linalg.eig(MT_M)

The results are the following:

$$v_{MM^T} = \begin{bmatrix} -0.1649 & -0.9554 & 0.2500 & -0.5400 & -0.7850 \\ -0.4716 & -0.0348 & -0.4533 & -0.6202 & 0.3029 \\ -0.3365 & 0.2708 & 0.8294 & -0.1270 & 0.2857 \\ -0.0033 & 0.04410 & 0.1697 & 0.1602 & 0.4371 \\ -0.7982 & 0.1037 & -0.1331 & 0.5310 & -0.1390 \end{bmatrix}$$

$$v_{M^TM} = \begin{bmatrix} 0.4262 & 0.9045 & -0.0146 \\ 0.6150 & -0.3015 & -0.7286 \\ 0.6634 & -0.3015 & 0.6848 \end{bmatrix}$$

(d) Find the SVD for the original matrix M from parts (b) and (c). Note that there are only two nonzero eigenvalues, so your matrix Σ should have only two singular values, while U and V have only two columns.

For the SVD, we can use the following properties:

- Our Σ matrix will be the diagonal matrix of the square roots of the non-zero eigen values
- Our U matrix will be the eigenvectors of MM^T that are associated with the non-zero eigenvalues of MM^T.
- Our V^T matrix will be the eigenvectors of M^TM that are associated with the non-zero eigenvalues of M^TM.
- Lastly, using the method for V^T described above led to a lot of issues with signs (negatives, positives, etc.). To fix this, we can use the following property to calculate V^T instead using our M, U, and Σ matrices: $V^T = \Sigma^{-1}U^TM$.

Using these principles, we can say that our Σ matrix is :

$$\Sigma = \begin{bmatrix} \sqrt{214.67} & 0 \\ 0 & \sqrt{69.33} \end{bmatrix}$$

This is because from both eigen value vectors, we see that these are the two non zero eigen values in this order, and we must take the square root of them for the Σ matrix. In λ_{MM^T}, these eigen values are at the 0th and 2nd index. Using our principles above, that means that U will be comprised of the two columns at the 0th and 2nd indices of v_{MM^T}:

$$U = \begin{bmatrix} -0.1649 & 0.2500 \\ -0.4716 & -0.4533 \\ -0.3365 & 0.8294 \\ -0.0033 & 0.1697 \\ -0.7982 & -0.1331 \end{bmatrix}$$

In λ_{M^TM}, the non-zero eigen values are also at the 0th and 2nd index. Therefore, V will be comprised of the two columns at the 0th and 2nd indices of v_{M^TM} and V^T will simply be the transpose:

$$V = \begin{bmatrix} 0.4262 & -0.0146 \\ 0.6150 & -0.7286 \\ 0.6634 & 0.6848 \end{bmatrix}$$

$$V^T = \begin{bmatrix} 0.4262 & 0.6150 & 0.6634 \\ -0.0146 & -0.7286 & 0.6848 \end{bmatrix}$$

However, as discussed earlier, this method of calculating V^T, while conceptually sound, leads to some issues when reconstructing M, so we use $V^T = \Sigma^{-1}U^TM$ to calculate our V^T instead, which gives us:

$$V^T = \begin{bmatrix} -0.4262 & -0.6150 & -0.6634 \\ -0.0146 & -0.7286 & 0.6848 \end{bmatrix}$$

Therefore, our final SVD decomposition matrices are as follows:

$$U = \begin{bmatrix} -0.1649 & 0.2500 \\ -0.4716 & -0.4533 \\ -0.3365 & 0.8294 \\ -0.0033 & 0.1697 \\ -0.7982 & -0.1331 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 14.65 & 0 \\ 0 & 8.33 \end{bmatrix}$$

$$V^T = \begin{bmatrix} -0.4262 & -0.6150 & -0.6634 \\ -0.0146 & -0.7286 & 0.6848 \end{bmatrix}$$

If we multiply these 3 matrices together, we get:

$$U\Sigma V^T = \begin{bmatrix} 1 & 0 & 3 \\ 3 & 7 & 2 \\ 2 & -2 & 8 \\ 0 & -1 & 1 \\ 5 & 8 & 7 \end{bmatrix}$$

As we can see, this perfectly matches our M matrix, meaning that we properly decomposed our M matrix using SVD.

(e) Set your smaller singular value to 0 and compute the one-dimensional approximation to the matrix M.

Using the same principles as in part d, we start with our Σ matrix from part d. However, we set the second singular value to 0, so that there is now only one singular value. Therefore, our Σ matrix will just be the diagonal (1 by 1) matrix of this singular value:

$$\Sigma = \begin{bmatrix} \sqrt{214.67} \end{bmatrix}$$

From both eigen value vectors, we see that these are the two non zero eigen values. However, we set the second one to 0, so we must take the square root of only the first eigen value for the Σ matrix. In λ_{MM^T}, this eigen value is at the 0th index. Using our principles above, that means that U will be comprised of the one column at the 0th index of v_{MM^T}:

$$U = \begin{bmatrix} -0.1649 \\ -0.4716 \\ -0.3365 \\ -0.0033 \\ -0.7982 \end{bmatrix}$$

In λ_{M^TM}, the non-zero eigen value is also at the 0th index. Therefore, V will be comprised of the one column at the 0th index of v_{M^TM} and V^T will simply be the transpose:

$$V = \begin{bmatrix} 0.4262 \\ 0.6150 \\ 0.6634 \end{bmatrix}$$

$$V^T = \begin{bmatrix} 0.4262 & 0.6150 & 0.6634 \end{bmatrix}$$

However, as discussed earlier, this method of calculating V^T, while conceptually sound, leads to some issues when reconstructing M, so we use $V^T = \Sigma^{-1}U^TM$ to calculate our V^T instead, which gives us:

$$V^T = \begin{bmatrix} -0.4262 & -0.6150 & -0.6634 \end{bmatrix}$$

Therefore, our final SVD decomposition matrices are as follows:

$$U = \begin{bmatrix} -0.1649 \\ -0.4716 \\ -0.3365 \\ -0.0033 \\ -0.7982 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 14.65 \end{bmatrix}$$

$$V^T = \begin{bmatrix} -0.4262 & -0.6150 & -0.6634 \end{bmatrix}$$

If we multiply these 3 matrices together, we get:

$$U\Sigma V^T = \begin{bmatrix} 1.0298 & 1.4862 & 1.6032 \\ 2.9449 & 4.2500 & 4.5847 \\ 2.1009 & 3.0319 & 3.2707 \\ 0.0206 & 0.0298 & 0.0321 \\ 4.9838 & 7.1925 & 7.7590 \end{bmatrix}$$

In this case, our reconstruction of M by multiplying our U, Σ, and V^T matrices (one dimensional approximation) appears much less accurate. The first column is almost nearly identical. However, the other 2 columns don't seem too close to the 2nd and 3rd columns of M. One interesting thing to notice is that the two columns seem to converge towards the same values such that they are almost equal. Another interesting observation is that adding the second and third columns together in our approximation is almost equivalent to adding the second and third matrices of M. So, while they aren't actually the same in value, there does seem to be a connection between the two matrices. Reducing to one singular value should be expected to produce such a result, as one singular value will most likely not be the best predictor.

2. Principal Components of Standardized Data. Recall that the principal components of uncentered data X are given by the eigenvectors of $(X - \bar{x})^T - (X - \bar{x})$ in descending order of corresponding eigenvalue, where $\bar{x}$ is the mean of the rows of X. The matrix $\hat{\Sigma} = \frac{1}{n}(X - \bar{x})^T(X - \bar{x})$ is known as the covariance matrix of X and it is always PSD. Suppose it is also invertible. Let $X^S = (X - \bar{x})\ \hat{\Sigma}^{-1/2}$ be the standardization of X – that is, a linear transformation of X so that each column has zero mean and unit variance and each two columns have zero covariance. Such standardization is a common preprocessing used to make any point cloud X look like a symmetric sphere around the origin. What are the principal components of the standardized data X^S? Explain why we get the result we get in terms of PCA finding the directions of largest variation and comment on the practical implications of this for PCA applied to data where each column may be in different units of scale.

We will begin by calculating the covariance matrix of X^S, which is $(X^S)^T(X^S)$. We will discuss this in more detail below. We substitute in the given values:

$$= ((X - \bar{x})\hat{\Sigma}^{-1/2})^T\ ((X - \bar{x})\hat{\Sigma}^{-1/2})$$

Next, we leverage the property that $(AB^T) = B^TA^T$:

$$= (\hat{\Sigma}^{-1/2})^T(X - \bar{x})^T\ (X - \bar{x})\hat{\Sigma}^{-1/2}$$

Knowing that $\hat{\Sigma} = \frac{1}{n}\ (X - \bar{x})^T(X - \bar{x})$, we can determine that $(X - \bar{x})^T(X - \bar{x}) = n\hat{\Sigma}$. We then substitute that into the equation:

$$= (\hat{\Sigma}^{-1/2})^T n\hat{\Sigma}(\hat{\Sigma}^{-1/2})$$

From here, we can multiply like terms for sigma and move n:

$$= n(\hat{\Sigma}^{-1/2})^T(\hat{\Sigma}^{1/2})$$

Now, assuming sigma is a diagonal matrix, we can remove the transpose from the first sigma term as it will remain the same value:

$= n(\hat{\Sigma}^{-1/2})(\hat{\Sigma}^{1/2})$

Lastly, we can see that multiplying the last two sigma terms results in the identity matrix I. Therefore, we end up with

=nI

Commentary:
As we see from the proof above, $(X^S)^T(X^S)$ equals nI, where I is the identity matrix. We begin by calculating $(X^S)^T(X^S)$ because we want to identify the covariance matrix to better understand the variance of each variable in the data set and thus how it will influence PCA. To better understand this, we first have to start with the the objective of PCA: to decrease the dimensionality of the data while preserving the variance of the data as much as possible. We want to preserve the variance so we can still extract insights from its behavior in a format that is easier to use. From here we can proceed to asking how then does PCA preserve variance as much as possible. To do so, PCA looks to identify the components that drive the greatest variability in the data to preserve them and discard other components that are less insightful for the exercise. And so, we want to find the components with largest variance for PCA to be effective.

Going through this exercise shows us that ultimately, if we standardize our data, the result of the covariance matrix is $(X^S)^T(X^S)$, which equals nI, where I is the identity matrix. Thus the covariance matrix has the same value along the diagonal of the matrix. As stated on minitab's website:

"In the covariance matrix in the output, the off-diagonal elements contain the covariances of each pair of variables. The diagonal elements of the covariance matrix contain the variances of each variable."*

Thus we end up seeing that the variance for each component is equal to all others. This gives the spherical shape. Furthermore, we must realize this is problematic, because PCA will no longer be effective in identifying the componenets with the largest variance as they are all equal now. If the directions within the sphere are all possible, PCA loses its value.

This brings us to our last point, which is that there is a difference between normalizing and standardizing data. In order to account for different scales for different components, it is critical to normalize the data to better level the magnitude of the scales of each component so that certain coefficients don't drive PCA just because their scale is larger. This is to say, it is not the same as standardizing, which will cause the variance of all components to equal one another.

*Link: https://support.minitab.com/en-us/minitab-express/1/help-and-how-to/modeling-statistics/regression/how-to/covariance/interpret-the-results/