

INSTRUCTIONS

- **Due:** Wednesday, April 27, 2022 at 11:59 PM EDT.
- **Format:** Complete this pdf with your work and answers. Whether you edit the latex source, use a pdf annotator, or hand write / scan, make sure that your answers (tex'ed, typed, or handwritten) are within the dedicated regions for each question/part. If you do not follow this format, **we will deduct 5% from your final assignment grade.**
- **How to submit:** Submit a pdf with your answers on Gradescope. Log in and click on our class 10-315, click on the appropriate *Written* assignment, and upload your pdf containing your answers. Don't forget to submit the associated *Programming* component on Gradescope if there is any programming required.
- **Policy:** See the course website for homework policies and Academic Integrity.

Name	
Andrew ID	
Hours to complete (both written and programming)?	

Q1. [24pts] Kernel PCA

(a) [4pts] PCA in high-dimensional feature spaces

Recall that for standard PCA, the principal components for a zero-centered dataset $\mathbf{X} \in \mathbb{R}^{N \times D}$ are found by eigen decomposition of its covariance matrix \mathbf{C} . (Note that N is the number of samples, and D is the number of features).

Let $\mathbf{x}_i \in \mathbb{R}^D$ be the i^{th} row of \mathbf{X} as a column vector, and the covariance matrix can be expressed as:

$$\mathbf{C} = \frac{1}{N} \sum_i^N \mathbf{x}_i \mathbf{x}_i^T.$$

The j^{th} eigenvector (principal component) \mathbf{u}_j and its corresponding eigenvalue λ_j can be found by solving,

$$\mathbf{C} \mathbf{u}_j = \lambda_j \mathbf{u}_j$$

Now let us consider a mapping $\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M$, which projects each original data point onto a higher dimensional space ($M > D$). This space is called **feature space**. It may be possible to obtain better dimensionality reduction when PCA is applied in the (nonlinear) feature space.

First assume that the new data points after projection are also zero-centered, meaning

$$\sum_i^N \phi(\mathbf{x}_i) = \mathbf{0}$$

and they have \mathbf{S} as the covariance matrix

$$\mathbf{S} = \frac{1}{N} \sum_{i=1}^N \phi(\mathbf{x}_i) \phi(\mathbf{x}_i)^T.$$

If we do standard PCA directly in the feature space by solving the following eigen decomposition problem,

$$\mathbf{S} \mathbf{v}_j = \lambda_j \mathbf{v}_j,$$

what could potentially be problematic? Please provide a brief explanation.

Hint: Think about the size of \mathbf{S} .

In the subsequent problems, we will derive an alternate solution using kernels where the PCA in feature space can be achieved by eigendecomposition of an $N \times N$ matrix, instead of the $M \times M$ covariance matrix \mathbf{S} .

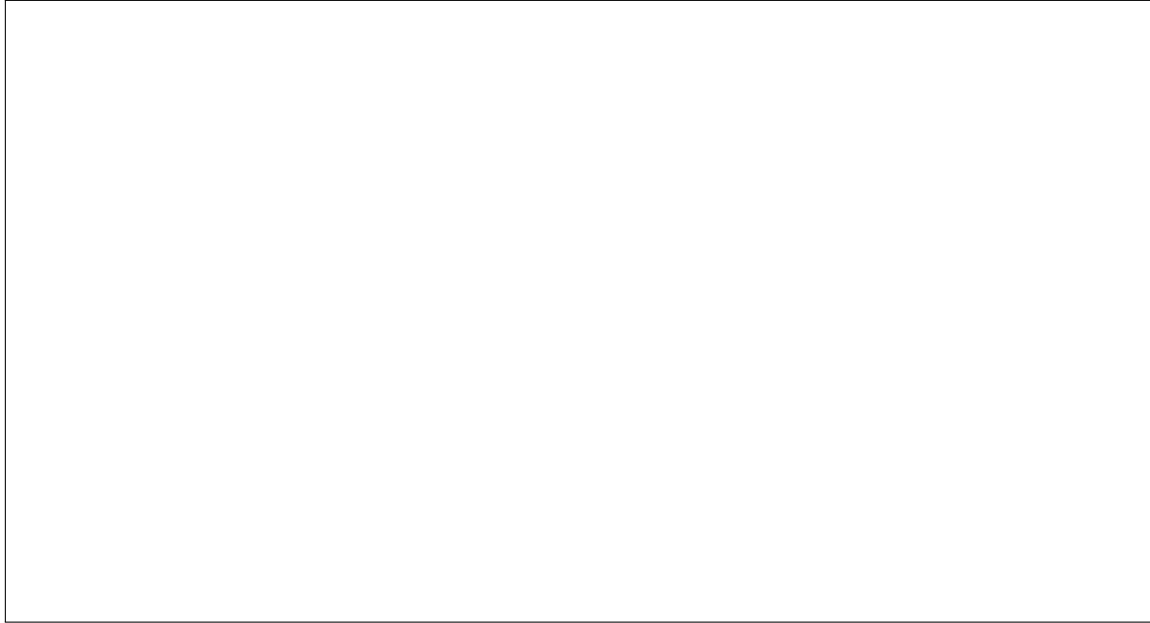
(b) [6pts]

Show that the j^{th} principal component \mathbf{v}_j can be expressed as a linear combination of transformed data points. That is, there exists an N -dimensional vector $\mathbf{w}_j = (w_{j1}, \dots, w_{jn}, \dots, w_{jN})^T$ such that:

$$\mathbf{v}_j = \sum_{i=1}^N w_{ji} \phi(\mathbf{x}_i) = \phi(\mathbf{X})^T \mathbf{w}_j$$

where $\phi(\mathbf{X}) = [\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \dots, \phi(\mathbf{x}_N)]^T$ is the $N \times M$ transformed data matrix.

Note: We only care about the eigenvalues that are non-zero.



Now you will show that the weight vector $\mathbf{w}_j, \forall j$ can be found by solving a $N \times N$ eigen decomposition problem. The idea here is to replace the covariance matrix \mathbf{S} with the kernel matrix

$$\mathbf{K} \in \mathbb{R}^{N \times N}, \mathbf{K}_{i,j} = k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j),$$

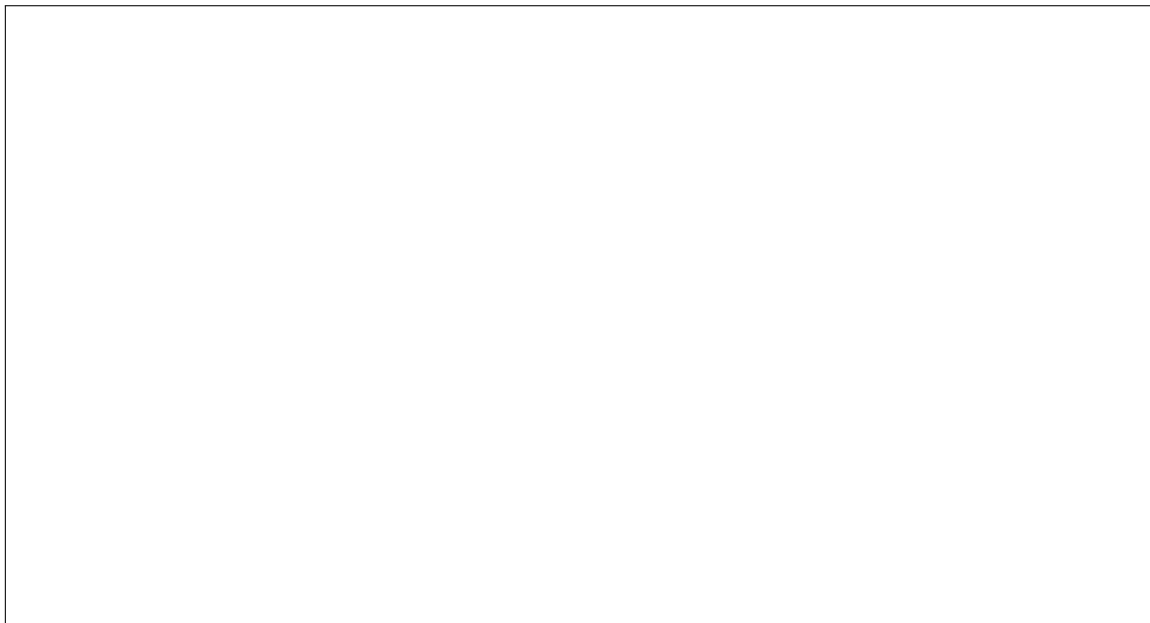
to avoid working in the feature space directly.

(c) [8pts]

Prove that any \mathbf{w}_j , of which the corresponding eigenvalue λ_j is non-zero, can be obtained by solving,

$$\mathbf{K} \mathbf{w}_j = N \lambda_j \mathbf{w}_j$$

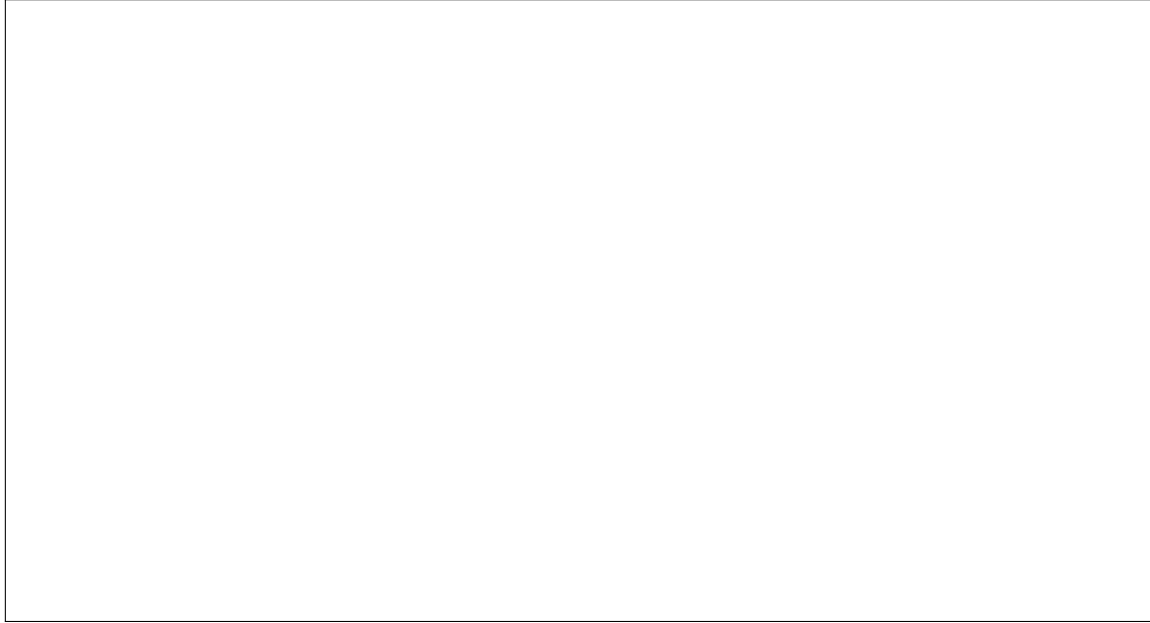
Hint: start from the original eigen decomposition problem in terms of \mathbf{S} and \mathbf{v}_j and use the results from the previous questions to arrive at the above equation. Also, you may assume \mathbf{K} is invertible.



(d) [6pts]

Notice that recovering the PC vector \mathbf{v}_j from the weight vector \mathbf{w}_j requires writing out the high-dimensional feature representation $\phi(\mathbf{X})$, which is problematic. In kernel PCA we avoid computing the PC vectors, and instead directly work with projections of data points onto the PC components. Show that the projection of any transformed point $\phi(\mathbf{x})$ onto the PC \mathbf{v}_j can be computed using the kernel and \mathbf{w}_j only as:

$$\phi(\mathbf{x})^\top \mathbf{v}_j = [k(\mathbf{x}, \mathbf{x}_1) \ k(\mathbf{x}, \mathbf{x}_2) \ \dots \ k(\mathbf{x}, \mathbf{x}_N)] \mathbf{w}_j$$



Q2. [12pts] Conceptual questions

(a) Suppose we have three data points in 2-D space, $(1, 1)$, $(2, 2)$ and $(3, 3)$.

(i) [2pts] What is the first principal component?

(ii) [2pts] If we want to project the original data points into 1-D space using this principal component, what will be the variance of the projected data?

(iii) [2pts] Suppose we want to reconstruct the original 2-D data using our projected data from (b). What is the reconstruction error in this case? Justify your answer.

(b) [2pts] True or False: It is reasonable to pick the number of clusters K in K-means clustering by minimizing the sum of squared distances between the data points and the cluster centers. Explain your answer.

(c) [2pts] Recall that in the E-step of the EM algorithm, we “softly” assign each data point to the the clusters. Under what parameter setting will the soft assignment in the EM algorithm with a univariate GMM reduce to hard assignment (as we’ve discussed in K-means)?

(d) [2pts] True or False: The EM algorithm will always converge to the global minimum because each EM step will monotonically improve the likelihood. Explain your answer.

Q3. [46pts] Programming: PCA and GMM

Introduction This assignment includes an autograder for you to grade your answers on your machine. This can be run with the command:

```
python3.6 autograder.py
```

The code for this assignment consists of several Python files, some of which you will need to read and understand in order to complete the assignment, and some of which you can ignore. You can download and unzip all the code, data, and supporting files from `hw4_programming.zip`.

Files you will edit

`kmeans.py` Your code to implement K-means clustering algorithm.

`gmm.py` Your code to implement clustering with Gaussian mixture models.

Files you might want to look at

`util.py` Convenience methods to generate various plots that will be needed in this assignment.

`test_cases/Q*/*.py` These are the unit tests that the autograder runs. Ideally, you would be writing these unit tests yourself, but we are saving you a bit of time and allowing the autograder to check these things. You should definitely be looking at these to see what is and is not being tested. These test cases also generate plots related to the task; the plots are saved in a directory named `figures`. The autograder on Gradescope may run a different version of these unit tests.

Files you can safely ignore

`autograder.py` Autograder infrastructure code.

Files to Edit and Submit: You will fill in portions of `kmeans.py` and `gmm.py` during the assignment. You should submit this file containing your code and comments to the Programming component on Gradescope. Please do NOT change the other files in this distribution or submit any of our original files other than these files. Please do not change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder.

Report: Many of the sections in this programming assignment will contain questions that are not autograded. You will place the requested results in the appropriate locations within the PDF of the Written component of this assignment.

Evaluation: Your assignment will be assessed based on your code, the output of the autograder, and the required contents of in the Written component.

Academic Dishonesty: We will be checking your code against other submissions in the class for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. These cheat detectors are quite hard to fool, so please don't try. We trust you all to submit your own work only; please don't let us down. If you do, we will pursue the strongest consequences available to us.

Getting Help: You are not alone! If you find yourself stuck on something, contact the course staff for help. Office hours, recitation, and Piazza are there for your support; please use them. If you can't make our office hours, let us know and we will schedule more. We want these assignments to be rewarding and instructional, not frustrating and demoralizing. But, we don't know when or how to help unless you ask.

(a) [9pts] Question 1: K-means

In `kmeans.py`, implement the `kmeans` function to perform K-means clustering algorithm on the MNIST dataset. See function docstring for details.

As you'll see in the code, we initialize the cluster centers to the first K points in the dataset. While this isn't an ideal initialization, it helps remove any randomness in the autograder.

You may run the following command to run some unit tests on your Q1 implementation:

```
python3.6 autograder.py -q Q1
```

The autograder will also render your 784 dimensional K-means centers as images. It will save these plots as png files in a new directory named `figures`. You are required to include some of these figures as part of the written component of this assignment. See the written component for details about which images to include.

(b) [9pts] Question 2: Gaussian Mixture Models

In this question, we'll be testing our GMM skills on an MNIST dataset with just the zero and one digits. Of course, since this is an unsupervised clustering task, we don't have any of the class labels for this dataset. To make our probabilistic clustering easier to visualize, we'll preprocess the MNIST dataset with PCA to compress it down to a 2-dimensional feature space.

PCA

In `gmm.py`, implement the `pca` function to use PCA to project data onto the first K principal components. See function docstring for details.

You may use either `numpy.linalg.eig` or `numpy.linalg.svd` to compute your eigenvectors. We recommend `numpy.linalg.svd` as it returns sorted arrays. You may NOT use high-level PCA solvers, for instance `sklearn.decomposition.PCA`.

NOTE: Eigenvectors are not unique. They can be scaled and still be correct eigenvectors. To make the autograder happy, you must pass your eigenvector matrix through `gmm.consistent_scale_eigenvectors` before using them to transform your data.

You may run the following command to run some unit tests on your Q2 PCA implementation:

```
python3.6 autograder.py -t test_cases/Q2/test1.py
python3.6 autograder.py -t test_cases/Q2/test2.py
```

EM for GMM

We've already given you the high level structure for the EM algorithm for Gaussian mixture models in `gmm.learn_gmm()`, which alternates between: E) updating the conditional class probabilities given the data and the parameters and M) updating the parameters given the data and the conditional class probabilities. While you shouldn't edit the core content of this `learn_gmm()` function, you may want to uncomment some of the printing and plotting statements to help you better understand what is happening at each iteration.

We've also provided a function to compute PDF of a multivariate Gaussian, `gmm.gaussian_pdf()`, and a function to compute the log likelihood of the GMM, `gmm.log_likelihood()`, that we are trying to maximize:

$$l(\theta; \mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(N)}) = \sum_{i=1}^N \log p(\mathbf{x}^{(i)} | \theta) = \sum_{i=1}^N \log \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}^{(i)}; \mu_{\mathbf{k}}, \Sigma_{\mathbf{k}})$$

where $\pi_k = p(z_k^{(i)} = 1), \forall i$.

E-step

Implement the `update_Z()` function in `gmm.py`. See the function docstring for details.

For a fixed set of Gaussian mixture model parameters, $\theta^{(t)}$, update the probability that each point, $\mathbf{x}^{(i)}$, belongs to the cluster k :

$$p(z_k^{(i)} = 1 | \mathbf{x}^{(i)}, \theta^{(t)}) \leftarrow \frac{\pi_k^{(i)} \mathcal{N}(\mathbf{x}^{(i)}; \mu_{\mathbf{k}}^{(t)}, \Sigma_{\mathbf{k}}^{(t)})}{\sum_{j=1}^K \pi_j^{(t)} \mathcal{N}(\mathbf{x}^{(i)}; \mu_{\mathbf{j}}^{(t)}, \Sigma_{\mathbf{j}}^{(t)})}, \forall i, k$$

M-step

Implement the `update_parameters()` function in `gmm.py`. See the function docstring for details.

$$\begin{aligned} \pi_k^{(t+1)} &\leftarrow \frac{\sum_{i=1}^N P(z_k^{(i)} = 1 | \mathbf{x}^{(i)}, \theta^{(t)})}{N}, \forall k \\ \mu_{\mathbf{k}}^{(t+1)} &\leftarrow \frac{\sum_{i=1}^N P(z_k^{(i)} = 1 | \mathbf{x}^{(i)}, \theta^{(t)}) \mathbf{x}^{(i)}}{\sum_{i=1}^N P(z_k^{(i)} = 1 | \mathbf{x}^{(i)}, \theta^{(t)})}, \forall k \\ \Sigma_{\mathbf{k}}^{t+1} &\leftarrow \frac{\sum_{i=1}^N P(z_k^{(i)} = 1 | \mathbf{x}^{(i)}, \theta^{(t)}) (\mathbf{x}^{(i)} - \mu_{\mathbf{k}}^{(t+1)}) (\mathbf{x}^{(i)} - \mu_{\mathbf{k}}^{(t+1)})^T}{\sum_{i=1}^N P(z_k^{(i)} = 1 | \mathbf{x}^{(i)}, \theta^{(t)})}, \forall k \end{aligned}$$

You may run the following command to run some unit tests on your Q2 implementation:

```
python3.6 autograder.py -q Q2
```

The autograder will also save various plots of PCA and GMM results as png files in a new directory named **figures**. You are required to include some of these figures as part of the written component of this assignment. See the written component for details about which plots to include.

Submission

Complete all questions as specified in the above instructions. Then upload `kmeans.py` and `gmm.py` to Gradescope. Your submission should finish running within 20 minutes, after which it will time out on Gradescope.

Don't forget to include any request results in the PDF of the Written component, which is to be submitted on Gradescope as well.

You may submit to Gradescope as many times as you like. You may also run the autograder on your own machine to speed up the development process. Just note that the autograder on Gradescope will be slightly different than the local autograder. The autograder can be invoked on your own machine using the command:

```
python3.6 autograder.py
```

Note that running the autograder locally will not register your grades with us. Remember to submit your code when you want to register your grades for this assignment.

The autograder on Gradescope might take a while but don't worry: **so long as you submit before the deadline, it's not late.**

The following questions should be completed after you work through the programming portion of this assignment.

(c) [12pts] PCA

Include the plots of the toy dataset before and after running PCA with $K=2$.

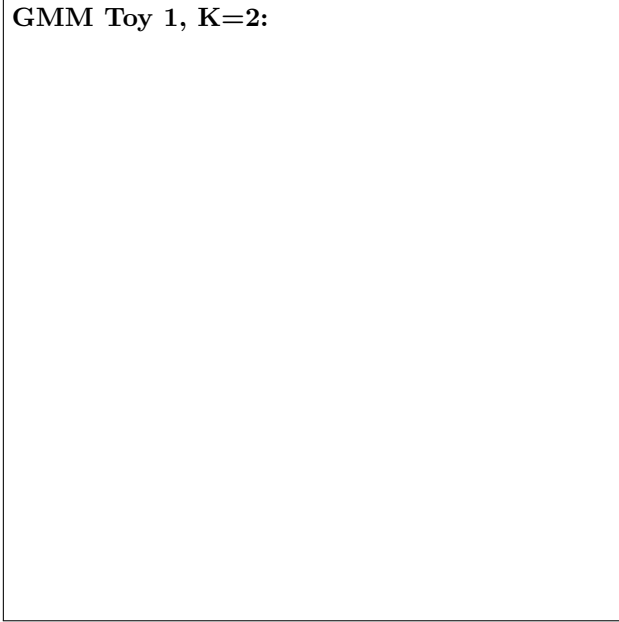
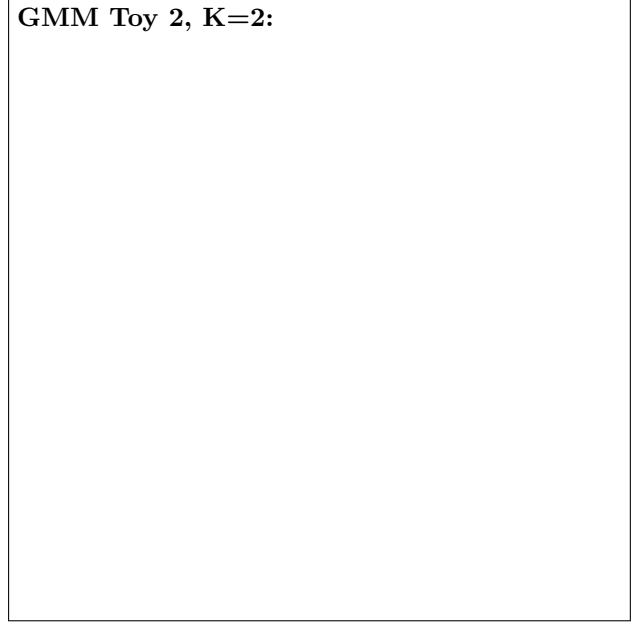
PCA before and after:

Include the plots of the MNIST zeros and ones dataset after running PCA with $K=2$.

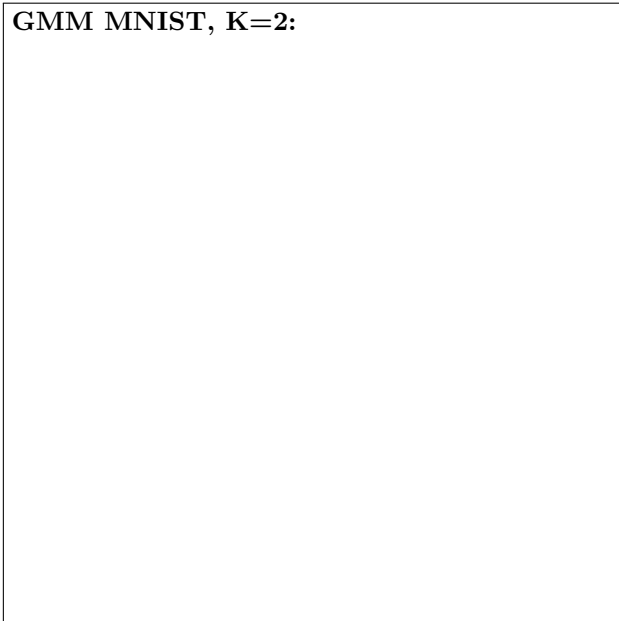
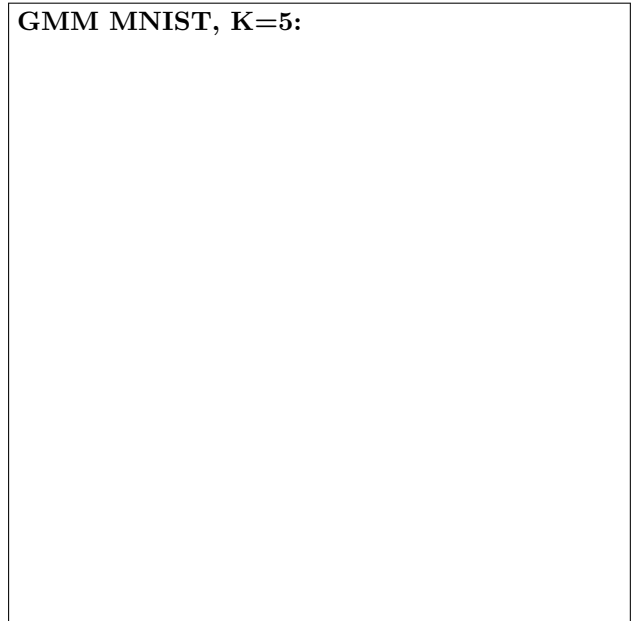
PCA MNIST:

(d) [8pts] GMM Toy Datasets

Include the plots after learning the GMM parameters for $K=2$ on toy dataset one and two.

GMM Toy 1, $K=2$:**GMM Toy 2, $K=2$:****(e)** [8pts] GMM MNIST Zeros and Ones

Include the plots after learning the GMM parameters for $K=2$ and $K=5$ on the MNIST zeros and ones dataset.

GMM MNIST, $K=2$:**GMM MNIST, $K=5$:**

Q4. [18pts] Programming: K-means

The following questions should be completed after you work through the programming portion of this assignment.

(a) [6pts] K=2

Include the images of the cluster centers after running k-means with two clusters.

Centers K=2:



(b) [6pts] K=5

Include the images of the cluster centers after running k-means with five clusters.

Centers K=5:



(c) [6pts] K=10

Include the images of the cluster centers after running k-means with ten clusters.

Centers K=10:

