ECE521 Assignment 3: Unsupervised Learning and Probabilistic Models

March 19, 2017

Raymond Ly (999959497)
Jianwei Sun (1000009821)
Wenyi Yin (1000084981)

1. K-means

1.1.1

The minimum function is not a convexity-preserving operation. For instance, consider the following example of scalar convex parabolic functions:

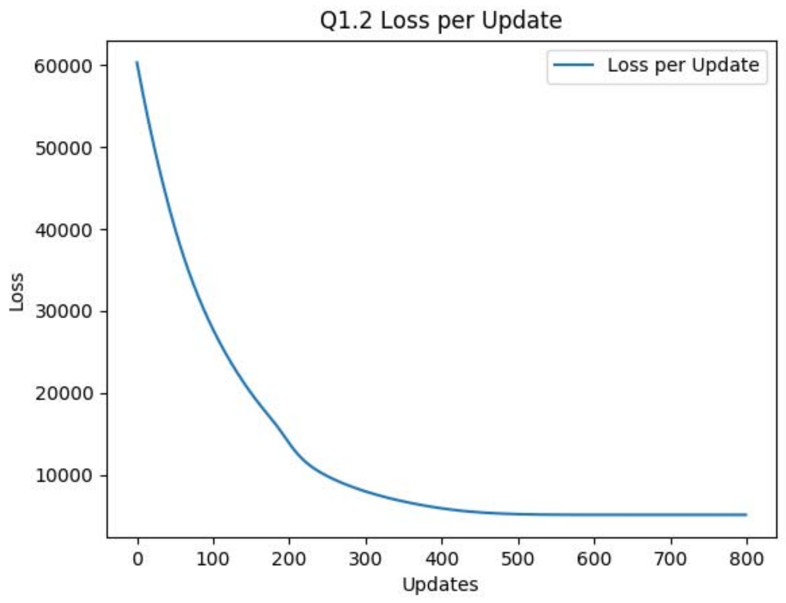
$$f(x) = min\{(x-1)^2,\ (x+1)^2\}$$

At $x = 0,\ f(0) = 1$. However, for $f(x)$ to be convex, the line segment joining $(-1, 0)$ and $(1,\ 0)$ must be above the function in this domain, according to Jensen's Inequality. Clearly, $x = 0$ is a counter-example as we have just shown. This example illustrates that the minimum function is not a convexity-preserving operation. As a result, since the variable μ is inside the minimum function in $L(\mu)$, the loss function is not necessarily convex, even if it is a summation of norms, which otherwise would have been convex.

1.1.2

In this question, the parameters, μ, were initialized by sampling from the standard normal distribution $N(0,\ 0.01^2)$. The loss per update was recorded and is plotted below.

Figure 1.1.2a Loss per update



The learning converged after a few hundred updates, as expected. However, a few hundred extra iterations were run to ensure convergence was reached.

1.1.3

For this question, the learning was run for five different values of K. After convergence, the data was coloured according to the cluster they belonged to, for visualization purposes.

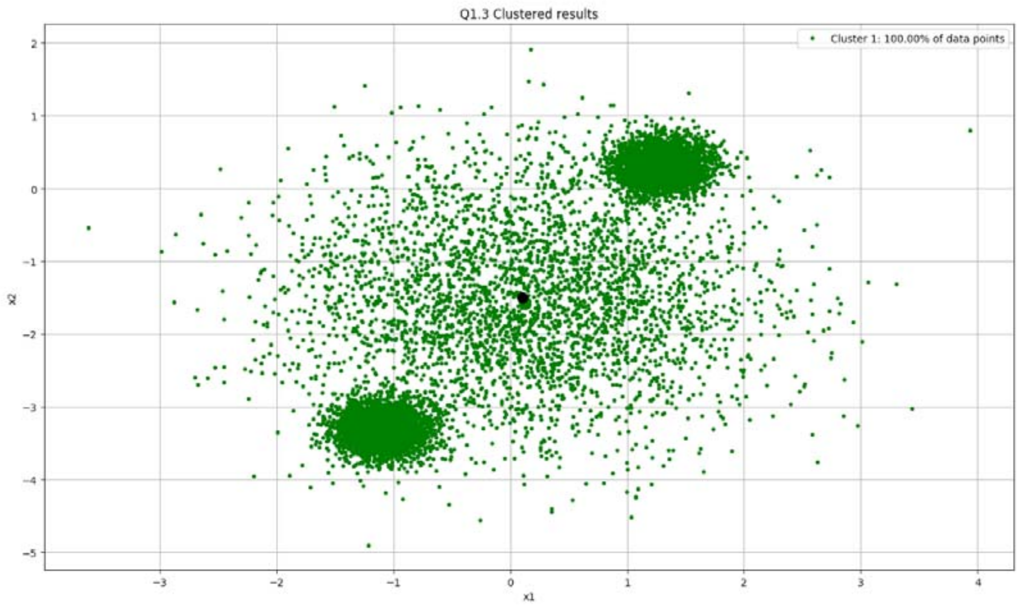


Figure 1.1.3a K = 1 Clustering

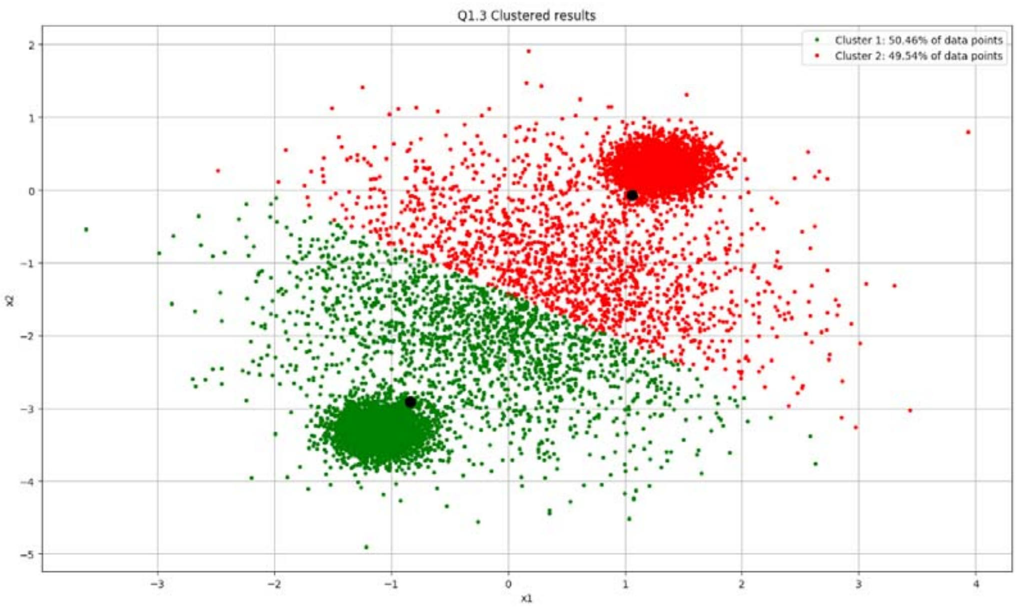


Figure 1.1.3b K = 2 Clustering

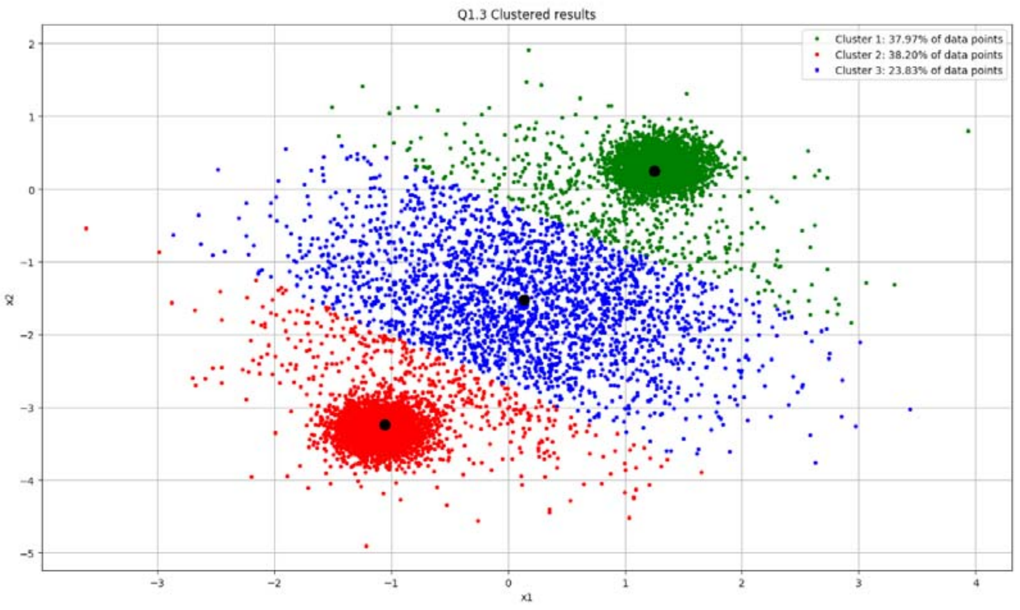


Figure 1.1.3c K = 3 Clustering

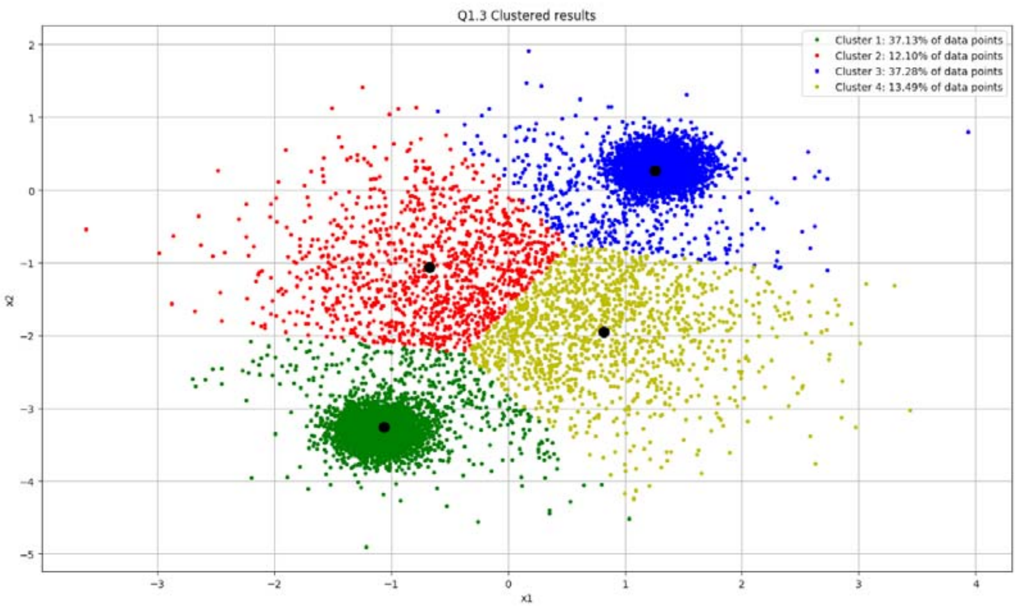


Figure 1.1.3d K = 4 Clustering

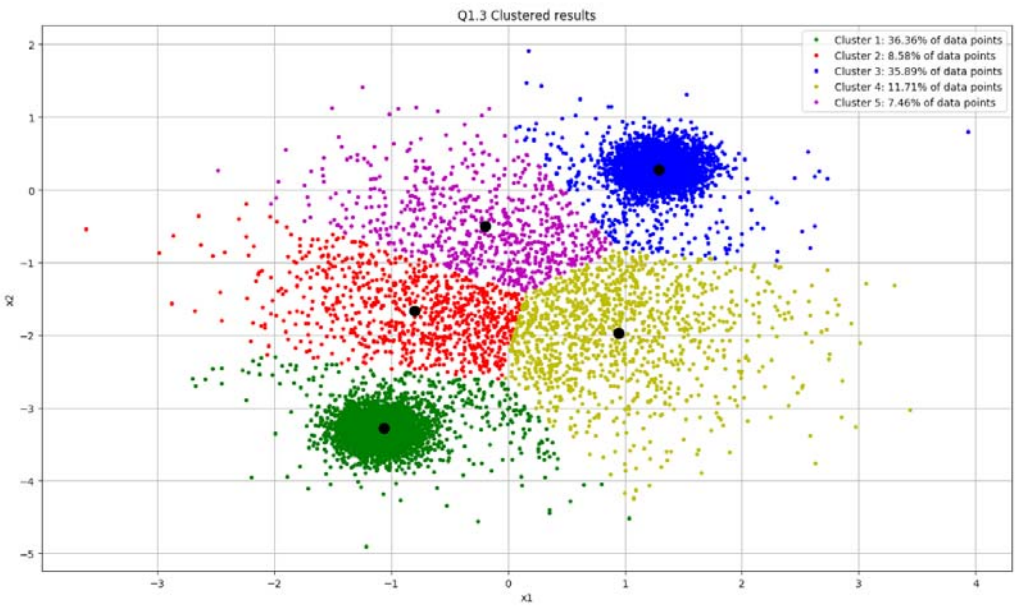


Figure 1.1.3e K = 5 Clustering

From the 2D scatter plots above, K = 3 produces the "best" clustering. Since there is no definition of what the "best" clustering is, we choose one that visually makes sense. The data appears to have two distinct clusters in the lower left and upper right corners, so it is expected that clusters would migrate to those centers. In addition, the rest of the data points appear to be scattered between the two clusters, with data becoming more sparse the further it is from the center of the line segment joining the two clusters. Hence, we intuitively expect K = 3 to capture the underlying structure of the data.

1.1.4

A third of the data was randomly held out for validation, and the remaining two thirds were used for training. The reported losses are tabulated below:

Table 1.1.4a. Loss Values after 800 Epochs for Various Number of Clusters

K	Loss Value
1	12877.4
2	3140.12
3	1744.86
4	1132.77
5	977.345

Solely based on these loss values, it appears that the best K in this set is K = 5 because it results in the lowest loss. It makes sense that a larger value of K would produce lower loss because there are more cluster centers to capture the structure of the data. As K increases to be equal to the number of data points, the loss would be zero since every cluster center would be at each data point. To prevent the model from overfitting, validation is used. However, in this case, the K is small, so even the validation set, which has the same underlying structure as the training set, suggests that K = 5 produces the best results.

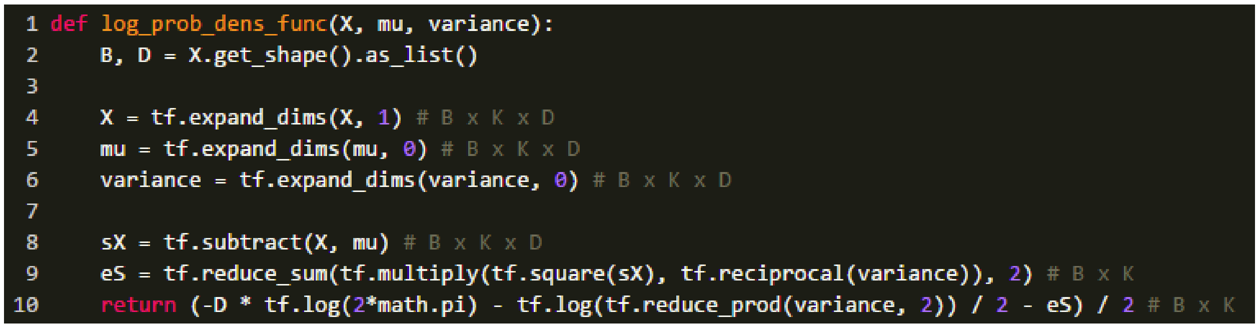
1. Mixtures of Gaussians

2.1.1

$$
\begin{aligned}
P(z=k \mid x) &= \frac{P(x, z=k)}{\sum_{j=1}^{K} P(x, z=j)} \\
&= \frac{P(z=k)\, P(x \mid z=k)}{\sum_{j=1}^{K} P(z=j)\, P(x \mid z=j)} \\
&= \frac{\pi^k\, N(\mu^k, \sigma^k)}{\sum_{i=1}^{K} \pi^i\, N(\mu^i, \sigma^i)}
\end{aligned}
$$

2.1.2

The log probability density function for cluster k, for all B data points and K clusters is computed with the following Python code:

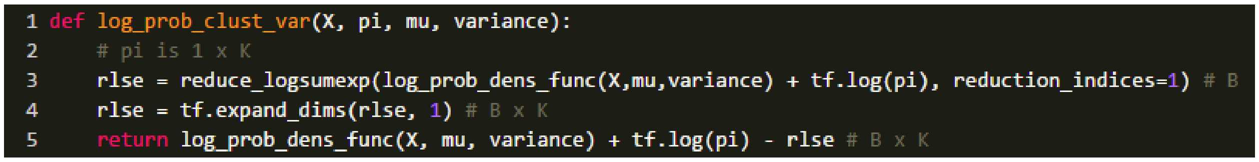


```python
1  def log_prob_dens_func(X, mu, variance):
2      B, D = X.get_shape().as_list()
3
4      X = tf.expand_dims(X, 1) # B x K x D
5      mu = tf.expand_dims(mu, 0) # B x K x D
6      variance = tf.expand_dims(variance, 0) # B x K x D
7
8      sX = tf.subtract(X, mu) # B x K x D
9      eS = tf.reduce_sum(tf.multiply(tf.square(sX), tf.reciprocal(variance)), 2) # B x K
10     return (-D * tf.log(2*math.pi) - tf.log(tf.reduce_prod(variance, 2)) / 2 - eS) / 2 # B x K
```

Lines 4 to 6 expands the dimensions of the input variables explicitly, which makes it clear to TensorFlow how to broadcast the data. Then, lines 8 to 10 compute the log of the multivariate Gaussian with diagonal covariance matrix in three steps.

2.1.3

The follow vectorized Python code computes the log probability of the cluster variable z given the data vector x. The code makes use of the previous log Gaussian pdf function and uses the logsumexp function provided by utils.py.



```python
def log_prob_clust_var(X, pi, mu, variance):
    # pi is 1 x K
    rlse = reduce_logsumexp(log_prob_dens_func(X,mu,variance) + tf.log(pi), reduction_indices=1) # B
    rlse = tf.expand_dims(rlse, 1) # B x K
    return log_prob_dens_func(X, mu, variance) + tf.log(pi) - rlse # B x K
```

The function expands dimensions explicitly to allow broadcasting to behave as expected. In this function, it is important to use the logsumexp function instead of tf.reduce_sum because the two operations are fundamentally different. Using a logsumexp function decreases the effect of the summation since the summation is applied in the exponential domain. When the log is then applied, the magnitude of the summation effect is reduced since the logarithm is a slowly increasing function. Further, since the probability density function for cluster k is provided as a log, we have to first exponentiate it before summation over all k clusters, which results in the need for logsumexp as opposed to just tf.reduce_sum.

2.2.1

$$LS = \nabla_\mu log P(x)$$

$$= \frac{1}{P(x)} \nabla_\mu \sum_k P(x|z=k)P(z=k)$$

Using the product rule for differentiation:

$$= \frac{1}{P(x)} \sum_k \left[P(x|z=k)\nabla_\mu P(z=k) + P(z=k)\nabla_\mu P(x|z=k)\right]$$

$$= \sum_k \left[\frac{P(x|z=k)}{P(x)}\nabla_\mu P(z=k) + \frac{P(z=k)}{P(x)}\nabla_\mu P(x|z=k)\right]$$

Then, by Bayes' Rule, we try to get the same numerator: $P(z=k|x)$

$$= \sum_k \left[\frac{P(z=k|x)}{P(z=k)}\nabla_\mu P(z=k) + \frac{P(z=k|x)}{P(x|z=k)}\nabla_\mu P(x|z=k)\right]$$

We notice that the previous result is achieved by moving the logarithm inside the del operator.

$$= \sum_k \left[P(z=k|x)\nabla_\mu log P(z=k) + P(z=k|x)\nabla_\mu log P(x|z=k)\right]$$

$$= \sum_k P(z=k|x)\left[\nabla_\mu log P(z=k) + \nabla_\mu log P(x|z=k)\right]$$

$$= \sum_k P(z=k|x)\ \nabla_\mu log P(x, z=k) = RS$$

2.2.2

For the data2D.npy dataset, K was set to be 3. The various constraints on standard deviation and π were satisfied using the suggested methods. After training, the best model parameters learnt can be seen in Table 2.2.2a below.

Table 2.2.2a. Final Model Parameters

Cluster	Mu	Variance	Log Pi
1	(-1.10459, -3.30753)	(0.07402, 0.07308)	-1.26571
2	(0.10220, -1.52518)	(2.19048, 3.02423)	-0.83566
3	(1.30116, 0.31063)	(0.07476, 0.07198)	-1.25747

The plot of the cluster centers along with the data points and their associated clustering can be seen in the figure below.

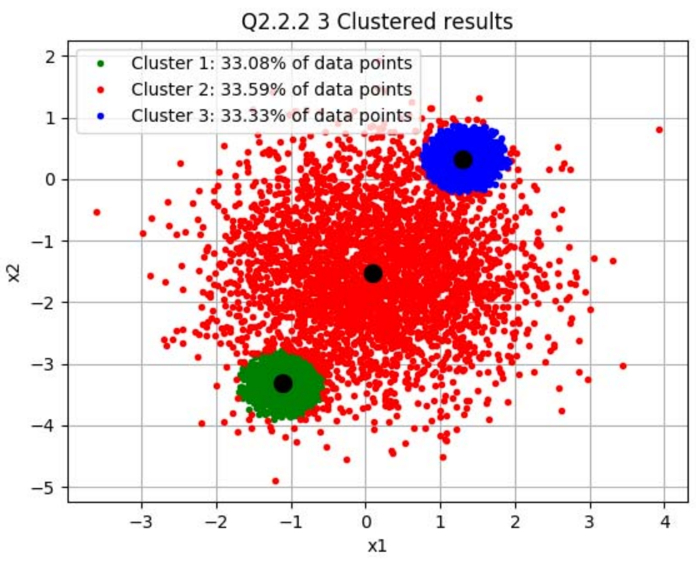


Figure 2.2.3a K = 1

Clusters 1 and 3 are more condensed and have a smaller standard deviations. This can be seen in both Figure 2.2.2a and Table 2.2.2a. The loss versus the number of updates reported is shown in the following plot. The final loss was 25833.8.

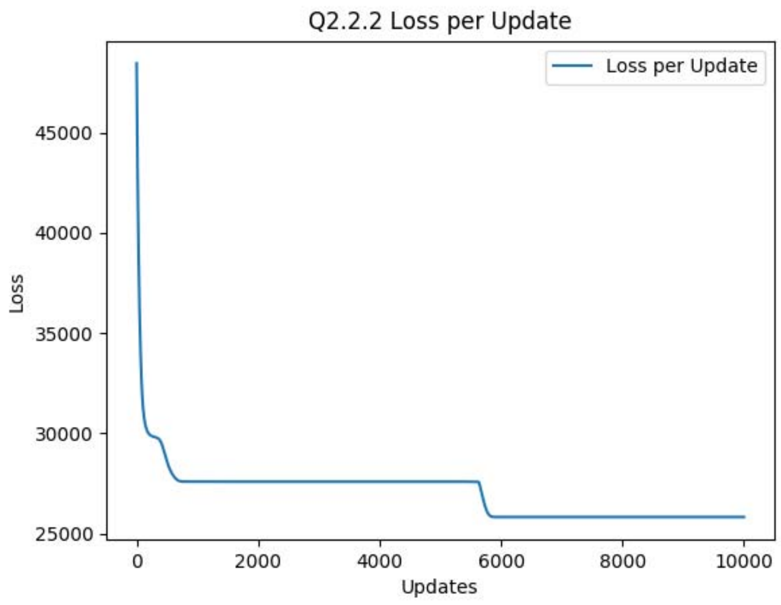


Figure 2.2.2a The negative log likelihood per update

2.2.3

In this question, a third of the data was randomly held out to act as validation data, and the remaining was used as training data. The five values of K were then used to train a Mixture of Gaussians model.

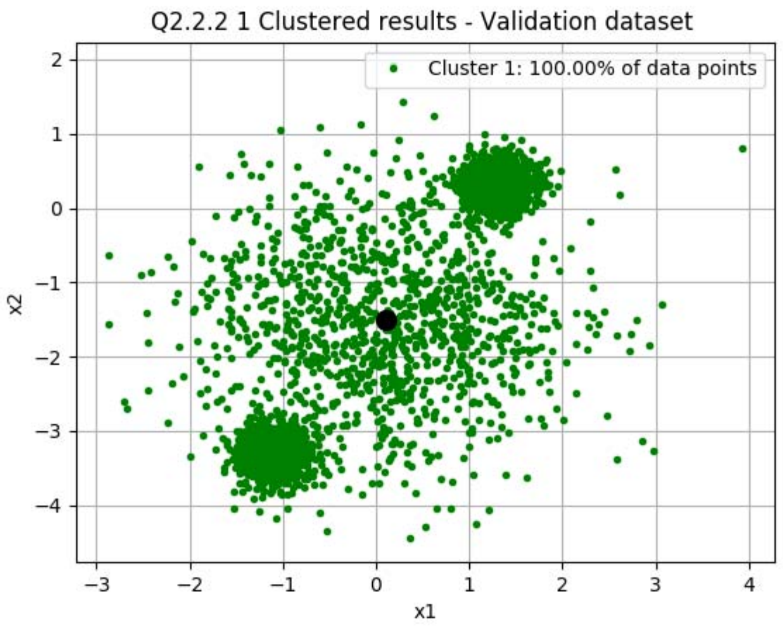


Figure 2.2.3a K = 1

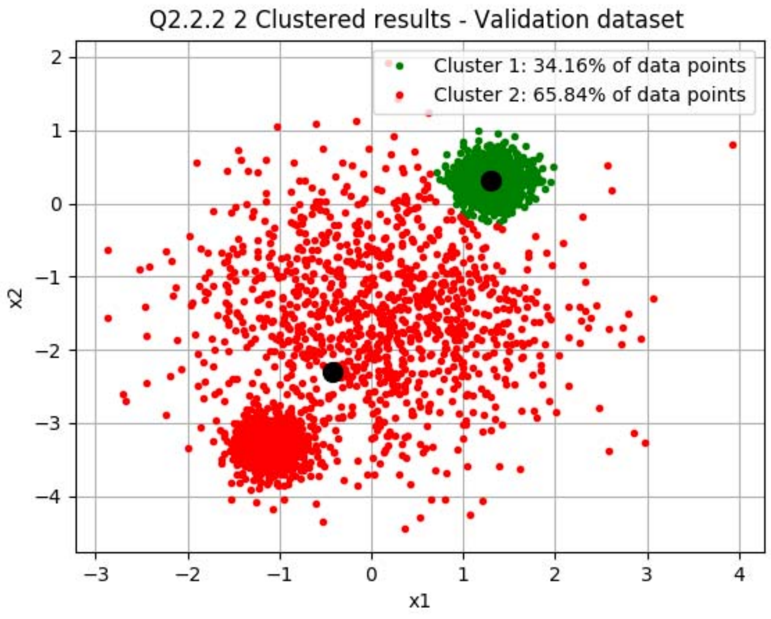


Figure 2.2.3b K = 2

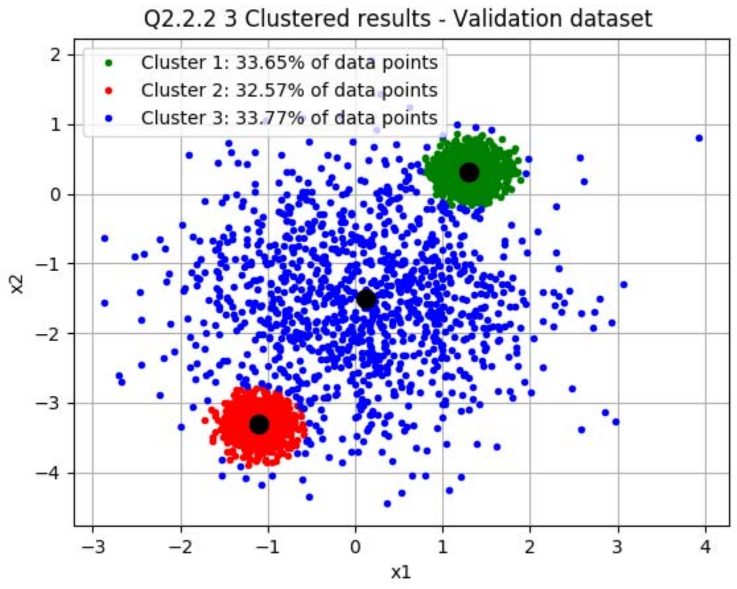


Figure 2.2.3c K = 3

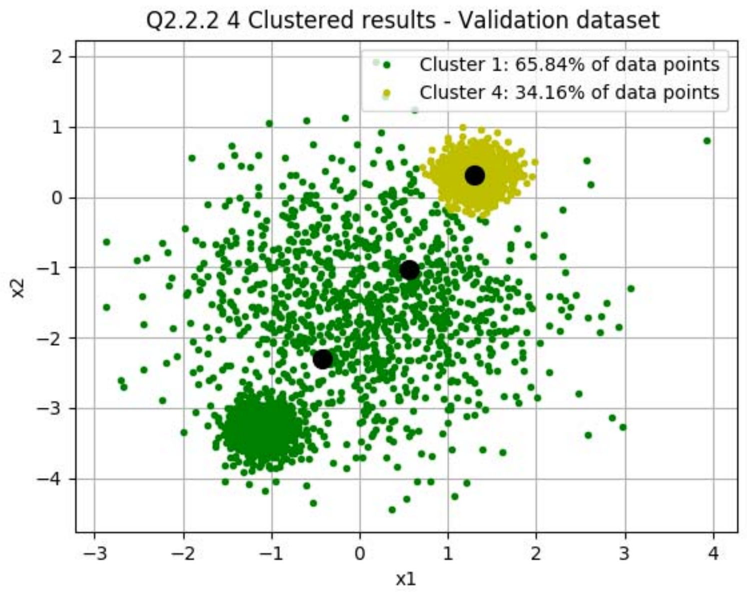


Figure 2.2.3d K = 4

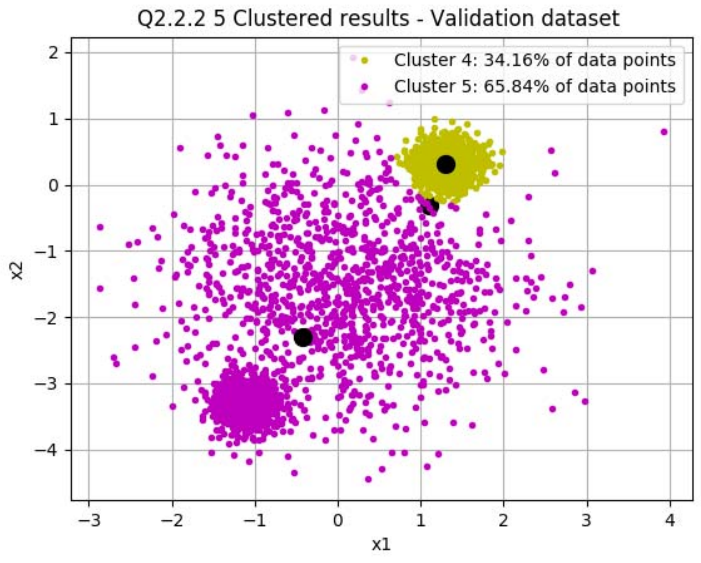


Figure 2.2.3e K = 5

In the plots for K = 4 and K = 5, not all of the clusters appear. During classification, the data points are assigned to clusters with the highest probability, so it is possible for some clusters to have a majority of points assigned to them, as in the case for K = 4 and K = 5.

The loss functions for the validation data for each of the five K values are tabulated below.

Table 2.2.3a. Loss Values for Various Number of Clusters

K	Loss Value
1	9960.68
2	9183.5
3	8629.25
4	9183.49
5	9183.54

From the above plots, it appears that K = 3 produces the visual results. Likewise, the loss value corresponding to K = 3 is the lowest among the five values of K, which supports K = 3 being the best choice. The data has an underlying structure that can be classified into three distinct groups, just based on visual inspection. Hence, it makes sense that the best value of K is one that matches these three distinct clusters.

2.2.4

K-means and MoG were run with K = 2, 5, 10, 15, 20. Below are the validation losses for each learning algorithm:

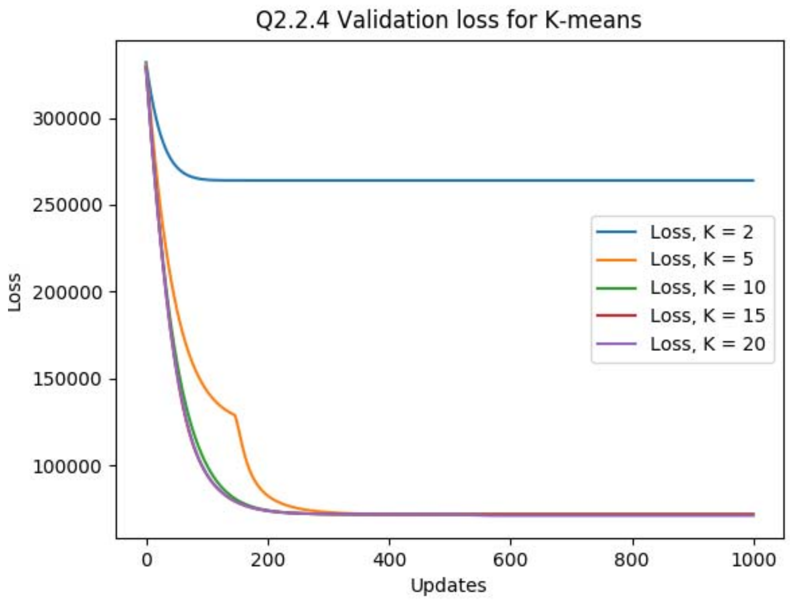


Figure 2.2.4a K-means validation loss over 1000 epochs.

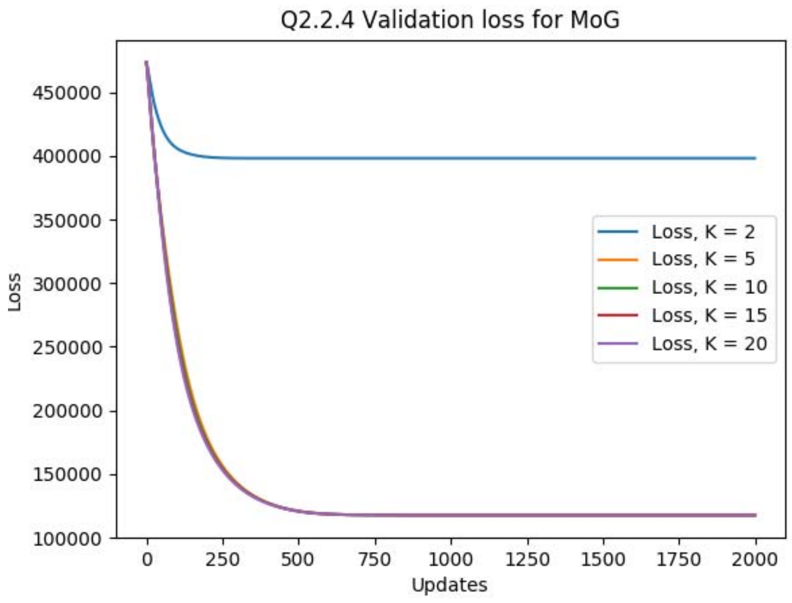


Figure 2.2.4b MoG validation loss over 2000 epochs.

From our observations, the 100D dataset appears to have approximately 5 clusters. The validation loss does not improve beyond K = 5, whereas the difference between 2 and 5 clusters is quite large. Looking at the cluster assignments for each model supports this statement - even with K = 10, no more than 5 clusters have data points assigned to them. See Table 2.2.4 below:

K = 2	K-means	MoG
Cluster 1	1616	1718
Cluster 2	1718	1616

K = 5	K-means	MoG
Cluster 1	953	953
Cluster 2	692	1381
Cluster 3	337	337
Cluster 4	663	663
Cluster 5	689	0

K = 10	K-means	MoG
Cluster 1	0	0
Cluster 2	692	692
Cluster 3	0	0
Cluster 4	953	953
Cluster 5	689	689
Cluster 6	0	0
Cluster 7	0	0
Cluster 8	663	663
Cluster 9	337	337
Cluster 10	0	0

Table 2.2.4 Cluster assignments for validation set, with various values for K.

The two models learned approximately the same cluster center locations for clusters 2, 4, 5, 8, and 9 as listed in Table 2.2.4 above, with distances in between these centers (between the two models) on the order of thousandths. The other cluster centers do not match as closely in between the two models, with nearest-distance on the order of 1-2.

3.1

We have $P(x) = \int_s P(x|s)P(s)ds$, where $P(x|s) = N(x;\ Ws + \mu,\ \Psi)$ and $P(s) = N(s;\ 0,\ I)$. Using the multivariate results (1), (2), and (3), we notice that $P(x|s)$ is exactly in the form of (2) and $P(s)$ is in the form of (1). Hence, simply applying (3), we get that $P(x) = N(x;\ W*0 + \mu,\ \Psi + WIW^T)$, which reduces to $P(x) = N(x;\ \mu,\ \Psi + WW^T)$.
Applying the logarithm to both sides, $log\ P(x) = log\ N(x;\ \mu,\ \Psi + WW^T)$, as required.

3.2

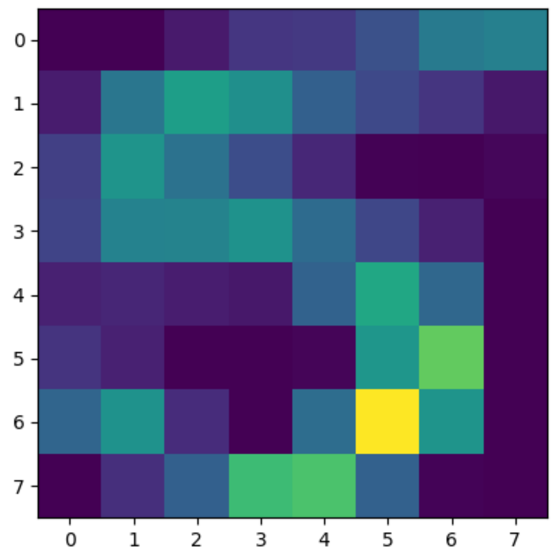


Figure 3.2a: Learnt Weight Matrix Visualization 1

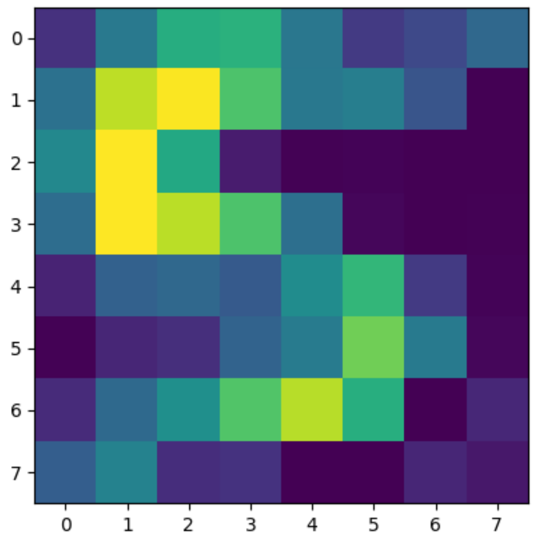


Figure 3.2b: Learnt Weight Matrix Visualization 2

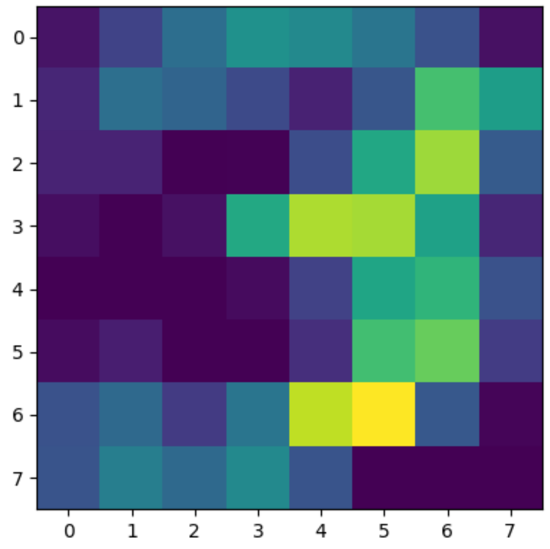


Figure 3.2c: Learnt Weight Matrix Visualization 3

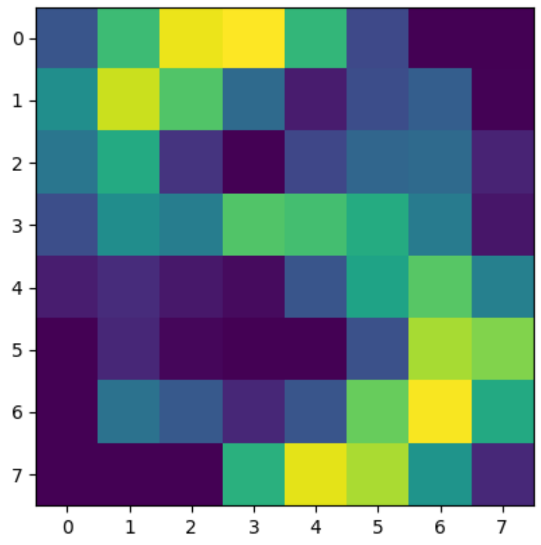


Figure 3.2d: Learnt Weight Matrix Visualization 4

Table 3.2a. Loss Values for Various Number of Clusters

Set	Log Likelihood
Validation	-4605.75
Test	-32191.0
Train	-18060.1

The test set and validation set log likelihoods were lower than the training set log likelihood, however, all three are in the same scale of magnitude. We aim to maximize the log likelihood on the training set which is in agreement with the results.

Each dimension of the weight matrix has captured a sense of variability about the handwritten digits of “3” and “5”. These figures have some resemblance to the digits 3 and 5 with some overlapping features in some of the visualizations. Some of the latent variables capture the similarities between the two data classifications while the others capture the differences. In Figure 3.2b, it can be seen that the weight matrix captures the leftmost straight edge of the ‘5’. This is a characteristic that is present only in the digit ‘5’. Another latent dimension, from Figure 3.2c, visualizes the right-middle point of the digit ‘3’ as well as the curve on the top right not present in the digit ‘5’ while placing low (negative) weighting to other features of ‘5’. One latent dimension, Figure 3.2d, encapsulates the top left and bottom right curves, a component that is common in both the digits. Finally, Figure 3.2a acquires just the bottom right curve.

3.3

The toy data set was generated from latent states, **s**, which was first generated from a 3D multivariate Gaussian distribution with zero mean and identity covariance matrix. To show that the single principal component learnt from PCA lines up with the maximum variance direction, the largest eigenvalue and eigenvectors were found from the covariance matrix of **x**. The maximum variance direction data was then plotting against the result of the projection of **x** onto the learnt single principal component.

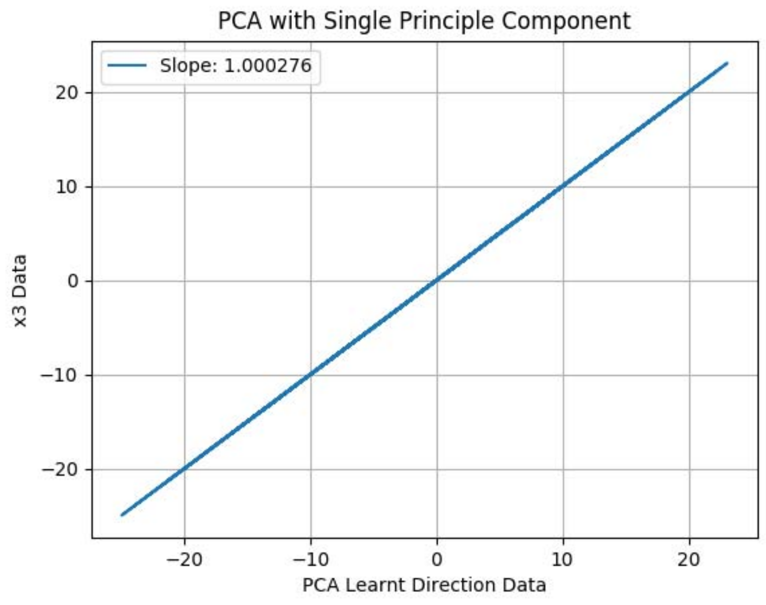


Figure 3.3a PCA results for a single principal component

From the diagram, a slope of unity shows that PCA has learnt the maximum variance direction, since it matches it very closely.

For the factor analysis model, a similar approach to the previous question was taken. The dataset **x** was projected onto the projection matrix given as below:

$$W_{proj} = (I + W^T \Psi^{-1} W)^{-1} W^T \Psi^{-1}$$

I is the identity matrix, W is the learnt weight matrix, and Ψ is the diagonal covariance matrix. The results of projecting **x** onto W_{proj} and then plotting it against the normalized $x_1 + x_2$ dataset is shown below.

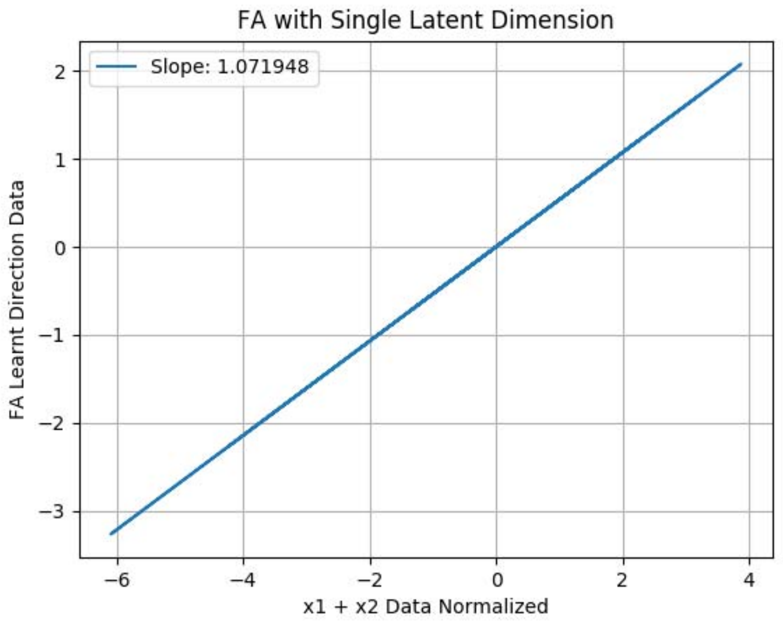


Figure 3.3b Factor Analysis results for a single latent dimension