ECE521: Assignment 3

Due on Tuesday, March 29, 2016

Davi Frossard
Erich Sato
Mauro Brito

March 29, 2016

K-Means

Part 1

We want an efficient, vectorized function of the pairwise distance between two matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$. To do so, let us first look at the case of a single entry.

$$\begin{aligned} d(x_i, y_i)^2 &= ||x_i - y_i||^2 \\ &= ||x_i||^2 + ||y_i||^2 - 2(x_i \cdot y_i^T) \end{aligned} \tag{1}$$

We can extend the function to the entire matrix with the concept of outer (element-wise) sum, for which we use the notation of $\dot{+}$. Therefore we extend Eq. 1.

$$\begin{aligned} d(X, Y)^2 &= ||X - Y||^2 \\ &= ||X||^2 \dot{+} ||Y||^2 - 2(X \cdot Y^T) \end{aligned} \tag{2}$$

With Eq. 2 and given matrices $\boldsymbol{X}$ with dimensions *MxN* and $\boldsymbol{Y}$ with dimensions *KxN*, we obtain a matrix $\boldsymbol{D}$ with dimensions *MxK* containing the pair-wise distance between the entries in $\boldsymbol{X}$ and $\boldsymbol{Y}$.

Part 2

We implement the pairwise squared distance discussed in Part 1 using Python and TensorFlow with the code listed in Code 1. To implement element-wise sum we use TensorFlow's broadcasting by operating with transpose of $||X||^2$ and $||Y||^2$, effectively computing the outer sum.

Code 1: Implementation of Pairwise Distance.

```python
def matrix_norm_column(x):
    '''
    Calculates the norm of each vector in a matrix, returning a column matrix
    '''
    x_sq = x*x
    norm = tf.reduce_sum(x_sq, 1, keep_dims=True)
    return norm

def square_distance(x, y):
    '''
    Returns pairwise squared distance between two matrices:
    For points xi belonging to X and yj belonging to Y, we have:
        d(xi,yj)^2 = ||xi - yj||^2 = ||xi||^2 + ||xj||^2 - 2 * (yj . xi)
    Expanding to the entire matrix, we get:
        d(X, Y)^2 = (||X||^2 .+ ||Y||^2) - 2*dot(Y,X')
            Where .+ is the outer (element-wise) sum
    '''

    x_norm = matrix_norm_column(x)
    y_norm = matrix_norm_column(y)
    dot_prod = tf.matmul(y, tf.transpose(x))

    outer_sum = tf.transpose(x_norm) + y_norm

    return outer_sum - 2 * dot_prod
```

Part 3

If we randomly initialize a dataset with 1000 points drawn from a normal distribution and evaluate the cost of K-Means with two cluster assuming one of a hundred points between -5 and 5, we get the cost function shown in Figure 1. From which we can see that the cost function is not convex.

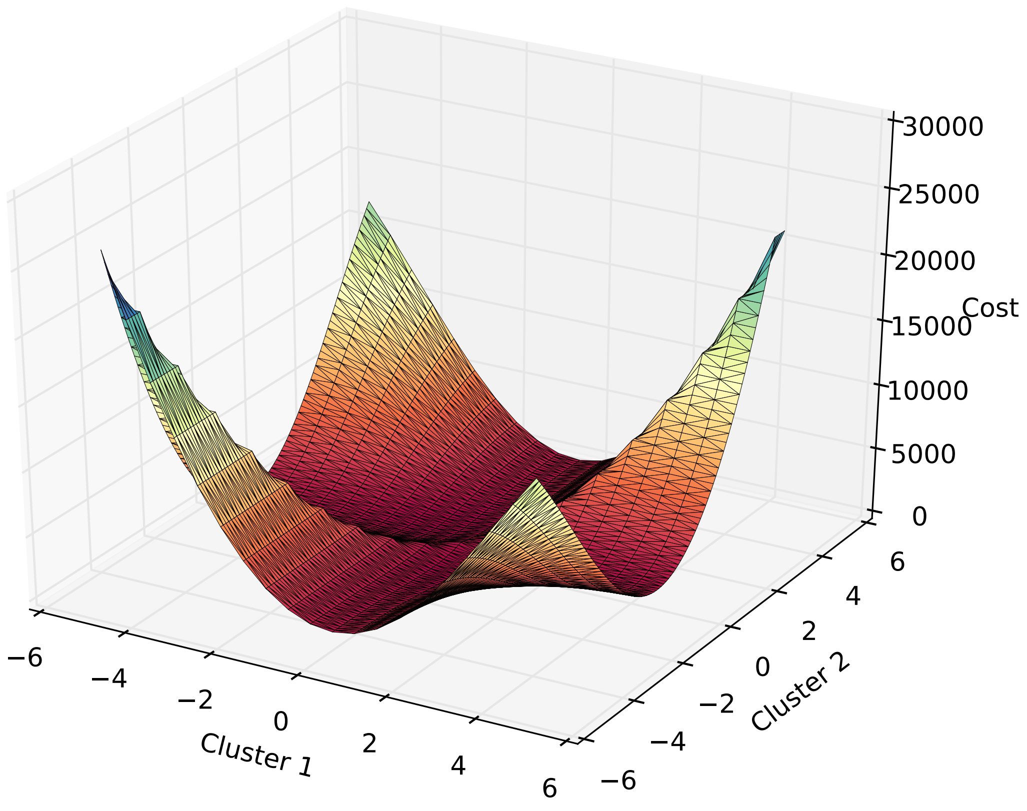

Figure 1: K-Means Cost Plot

We can generalize the argument following a simple argument for its non convexity: If we have any set of data X and cluster it into two optimal clusters K_1 with center C_1 and K_2 with center C_2, there is at least one other assignment that gives the same cost, it is K_1 with center C_2 and K_2 with center C_1. In fact, for higher numbers of clusters, any permutation of the centers will give the same cost.

Part 4

If we cluster the data provided in *data2D.npy* using K-Means with $K = 3$, we get the assignments shown in Figure 2. With the training curve shown in Figure 3.

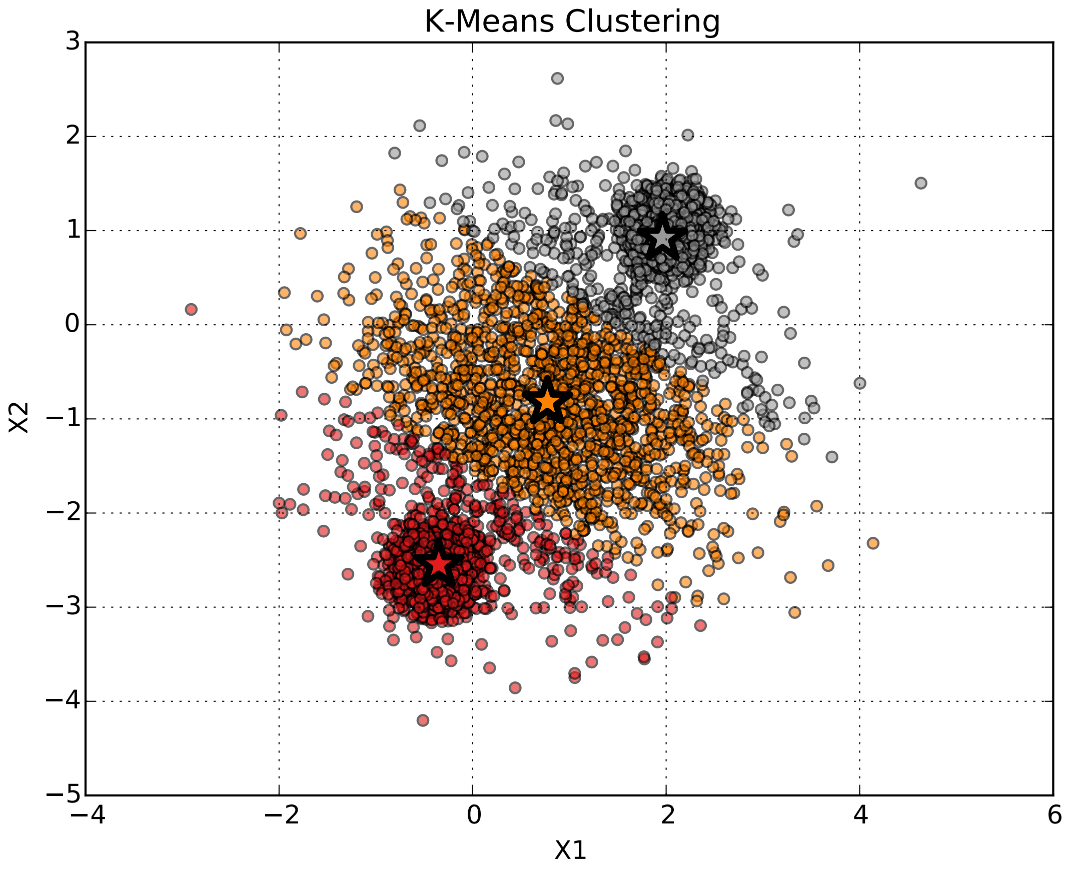

Figure 2: K-Means Cluster with $K = 3$

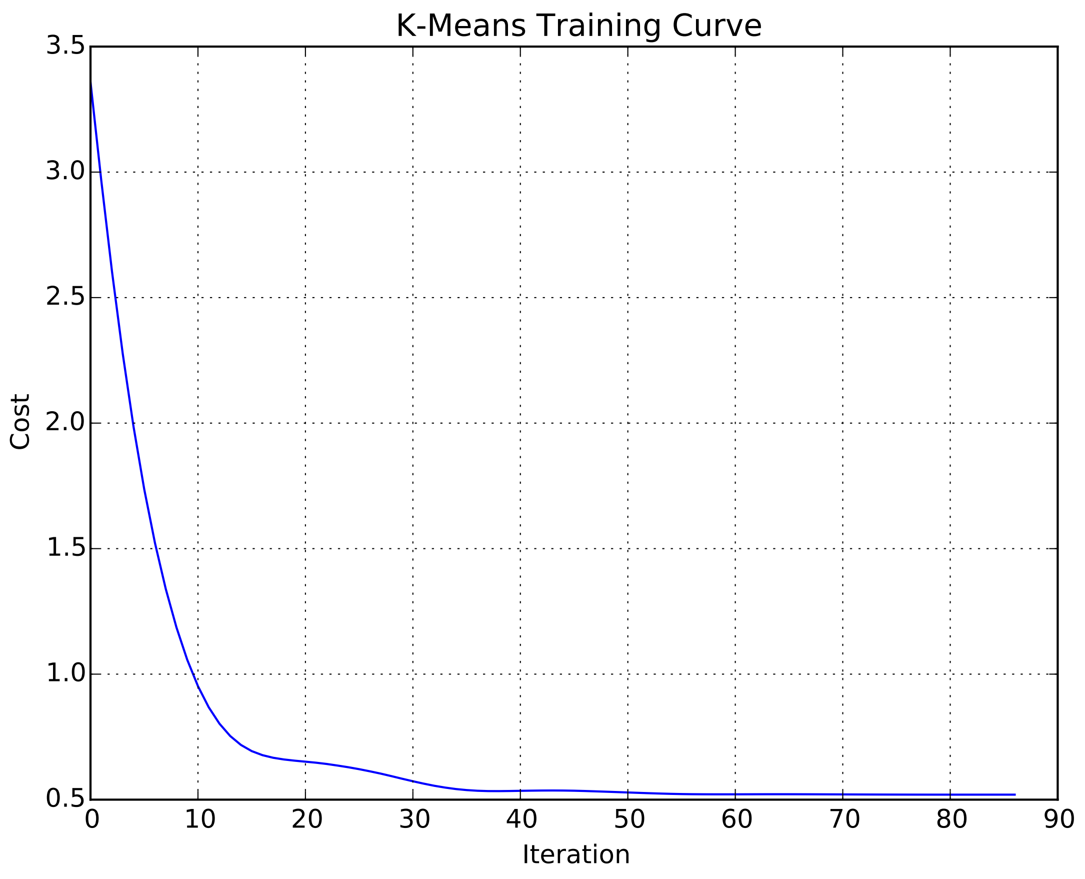

Figure 3: Training Curve

Part 5

Swiping the values of K from 1 to 5, we get the clusterizations depicted in Figure 4, with the percentages of data assigned to each cluster listed in Table 1.

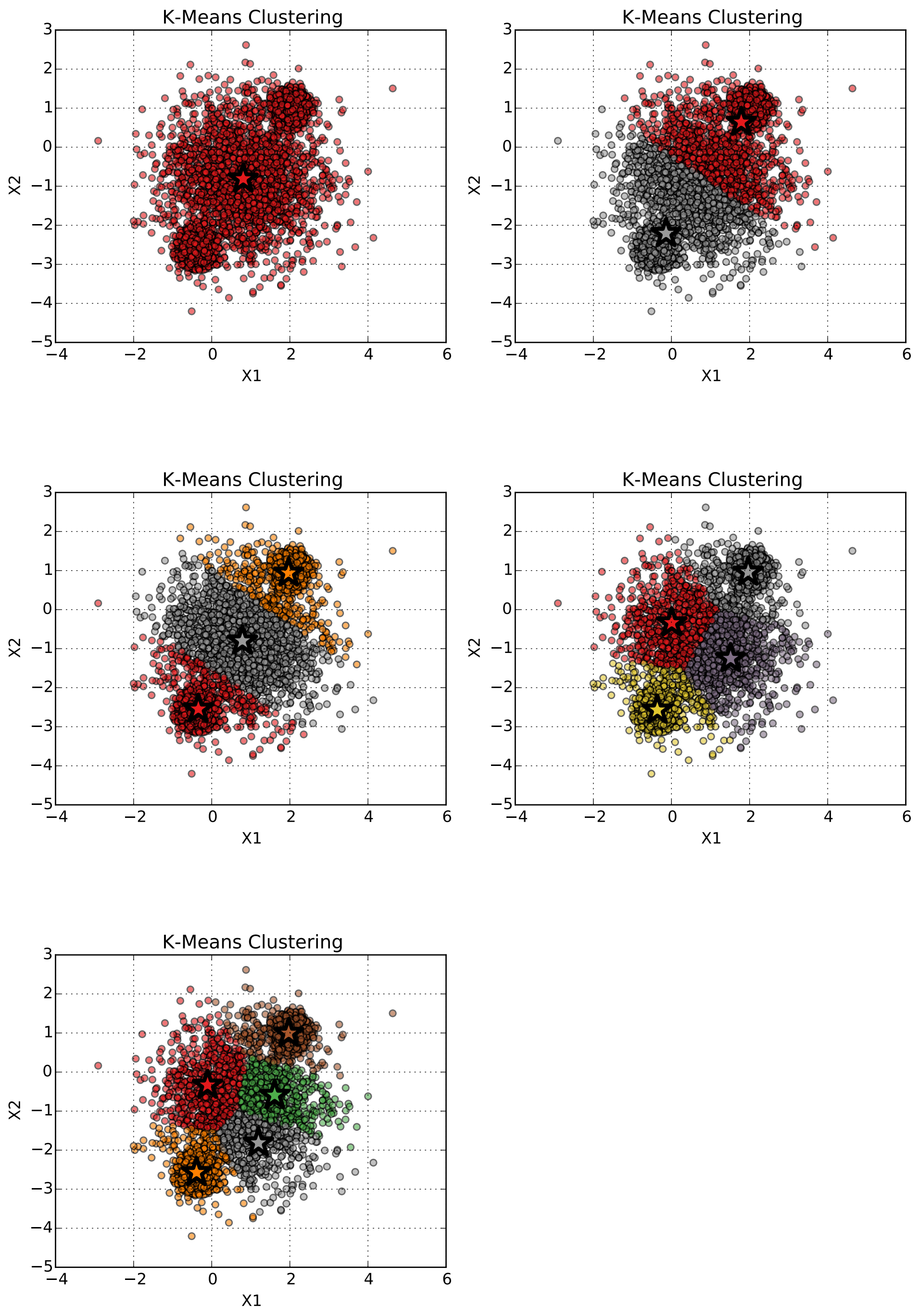

Figure 4: K-Means Clusterizations

	Percentage of Data in Cluster				
K	**1**	**2**	**3**	**4**	**5**
1	100.00	0.00	0.00	0.00	0.00
2	49.15	50.85	0.00	0.00	0.00
3	38.29	38.04	23.67	0.00	0.00
4	12.10	13.53	37.21	37.15	0.00
5	10.65	9.25	36.12	35.95	8.02

Table 1: Cluster Percentages.

Looking at table Table 1, we notice that increasing K up to 3 keeps the percentage of data in each cluster balanced. However, from $K = 4$ onwards we notice that some clusters are accumulating more data than others. From this perspective we conclude that $K = 3$ is the best parameter.

Part 6

Following an approach different of that from Part 5, we now separate a portion of the dataset for validation and calculate its loss for each value of K. We obtain Table 2

K	**Validation Cost**
1	3.826
2	0.907
3	0.495
4	0.332
5	0.282

Table 2: Validation costs for each value of K.

We notice that the validation cost is monotonically decreasing, so we expect that the larger the K, the smaller the cost. However, differently from supervised learning, we are not concerned with the cost itself, but rather with the rate of change. We notice that the factors of reduction in the validation cost decrease exponentially with the "elbow" of the curve at $K = 3$, as can also be seen in Figure 5, which reinforces our beliefs from Part 5.

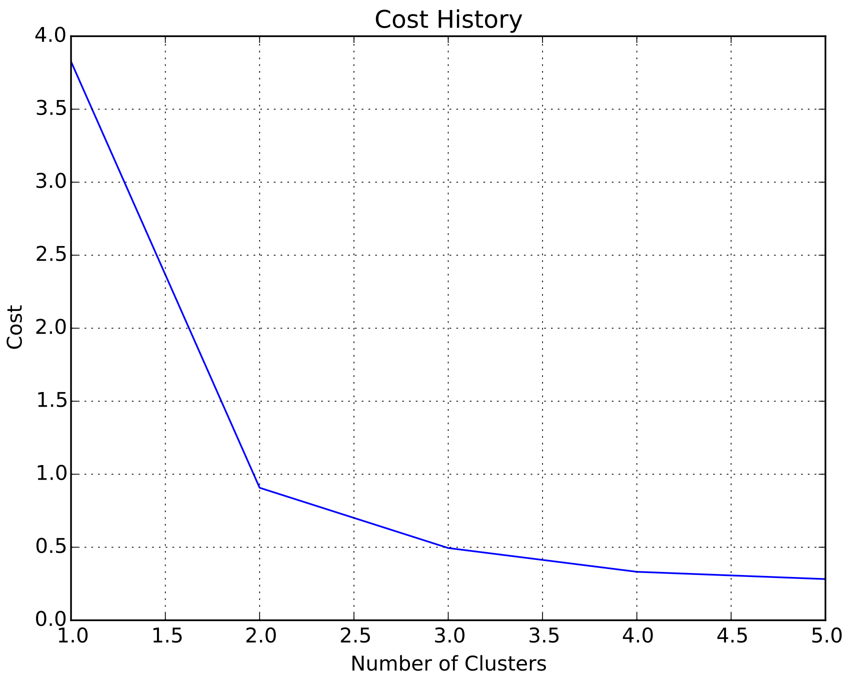

Figure 5: K-Means Clusterizations

Mixture of Gaussians

Part 1

We want to derive the expression for $P(z|\boldsymbol{x})$ in terms of the mixture of Gaussians parameters $\{\boldsymbol{\mu}_k, \sigma_k, \pi_k\}$:

$$\begin{aligned} P(z=k|\boldsymbol{x}) &= \frac{P(\boldsymbol{x}, z=k)}{\sum_{z'=1}^{K} P(\boldsymbol{x}|z')} \\ &= \frac{P(\boldsymbol{x}|z=k)P(z=k)}{\sum_{z'=1}^{K} P(\boldsymbol{x}|z')} \end{aligned} \tag{3}$$

We substitute $P(z=k)$ for the parameter π_k and extend the Gaussian distributions in terms of the parameters $\boldsymbol{\mu}_k$ and σ_k, thus obtaining Eq. 4.

$$P(z=k|\boldsymbol{x}) = \prod_{j=1}^{N} \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma_k} exp\left(-\frac{(x_j-\mu_k)^2}{2\sigma_k^2}\right)}{\sum_{l=1}^{K} \frac{1}{\sqrt{2\pi}\sigma_l} exp\left(-\frac{(x_j-\mu_l)^2}{2\sigma_l^2}\right)} \tag{4}$$

Part 2

Given cluster parameters $\boldsymbol{\mu_k}, \sigma_k^2$ we can find $log\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu_k}, \sigma_k^2)$ with Eq. 5

$$\begin{aligned} \mathcal{N}(x|\mu_k, \sigma_k^2)) &= P(x|\mu_k, \sigma_k^2) \\ log\mathcal{N}(x|\mu_k, \sigma_k^2)) &= logP(x|\mu_k, \sigma_k^2) \\ &= log\left(\frac{1}{\sqrt{2\pi}\sigma_k} exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)\right) \\ &= log\frac{1}{\sqrt{2\pi}\sigma_k} + \left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right) \end{aligned} \tag{5}$$

Moreover, we can expand the pairwise distance in the second term using our method from Part 1, we do so in Eq. 6.

$$log\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu_k}, \sigma_k^2)) = log\frac{1}{\sqrt{2\pi}\sigma_k} + \left(\frac{-(||x||^2. + ||\mu_k||^2 - 2\mu_k \cdot x^T)}{2\sigma_k^2}\right) \tag{6}$$

Our implementation of the equation derived in Eq. 6 is shown in Code 2.

Code 2: Implementation of Log Probability Density.

```python
def log_density(x, mu, sigma):
    '''
        Calculates log P(x | mu, sigma), i.e. the log probability density
        function for the mixture of gaussians
    Args:
        x: Data points
        mu: Cluster centers
        sigma: Cluster variance

    Returns:
        Log probability density function of the data
    '''
    den = tf.sqrt(2 * pi * sigma) # sqrt(2pi*sigma^2)
    D = tf.to_float(tf.rank(x))
    logp = -D*tf.log(den) # 1/(sqrt(2pi) * sigma)
    dists = square_distance(x, mu) # (x - mu)^2
    logp -= dists / (2*sigma) # 1/den - (x - mu)^2 / (2*sigma^2)

    return logp
```

Part 3

We now want to invert the probability found in Part 2: Given the data points and cluster means and standard deviations, find the probability of each cluster, i.e. $P(z|x)$.

$$\begin{aligned} P(z=k|x) &= \frac{P(x|z=k)P(z=k)}{P(x)} \\ &= \frac{P(x|z=k)P(z=k)}{\sum_{k'} P(x|z=k')P(z=k')} \end{aligned} \tag{7}$$

From Part 2, we have $logP(x|z=k)$, therefore we substitute in Eq. 7.

$$logP(z=k|x) = logP(x|z=k) + logP(z=k) - log\sum_{k'} exp\left[logP(x|z=k') + logP(z=k')\right] \tag{8}$$

We implement Eq. 8 (also generalizing for more than one training point) with Code 3.

Code 3: Implementation of Log Posterior Cluster Probability.

```python
def log_cluster_probability(x, logpz, mu, sigma):
    '''
        Computes the vector of log posterior cluster probabilities given the
        data, prior, mean and variance. P(Z | x)
    Args:
        x: Data
        pz: Prior probabilities
        mu: Gaussian mean
        sigma: Gaussian variance

    Returns:
        Vector of log posterior cluster probabilities
    '''

    logpxgz = log_density(x, mu, sigma) # logP(x | z)
    num = logpz + tf.transpose(logpxgz)
    den = reduce_logsumexp(num, 0)
    logpzgx = num/den # pz * P(x | z) / P(x)

    return logpzgx
```

Part 4

We want to show that $\nabla logP(x) = \sum_k P(z=k|x)\nabla logP(x,z=k)$. Let us first calculate $logP(x,z=k)$ with Eq. 9.

$$\begin{aligned}
P(x,z=k) &= P(z=k)P(x|z=k) \\
logP(x,z=k) &= logP(z=k) + logP(x|z=k) \\
&= log\pi_k + log\mathcal{N}(x|\mu_k,\sigma_k) \\
&= log\pi_k + log\frac{1}{\sqrt{2\pi}\sigma_k}exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right) \\
&= log\pi_k + log\frac{1}{\sqrt{2\pi}\sigma_k} + \left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)
\end{aligned} \tag{9}$$

We now calculate $P(x)$ with Eq. 10.

$$\begin{aligned}
P(x) &= \sum_k \pi_k\mathcal{N}(x|\mu_k,\sigma_k) \\
logP(x) &= log\sum_k \pi_k\frac{1}{\sqrt{2\pi}\sigma_k}exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)
\end{aligned} \tag{10}$$

We can now find $P(z=k|x)$ with Eq. 11.

$$\begin{aligned}
P(z=k|x) &= \frac{P(x|z=k)P(z=k)}{P(x)} \\
&= \frac{\pi_k\frac{1}{\sqrt{2\pi}\sigma_k}exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)}{\sum_l \pi_l\frac{1}{\sqrt{2\pi}\sigma_l}exp\left(\frac{-(x-\mu_l)^2}{2\sigma_l^2}\right)}
\end{aligned} \tag{11}$$

Now we calculate the derivative of Eq. 9 with respect to μ_k with Eq. 12.

$$\begin{aligned}
\nabla logP(x,z=k) &= \frac{\partial}{\partial\mu_k}\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right) \\
&= \frac{1}{2\sigma_k^2}\frac{\partial}{\partial\mu_k}\left((x-\mu_k)^2\right) \\
&= \frac{2}{2\sigma_k^2}(x-\mu_k) \\
&= \frac{x-\mu_k}{\sigma_k^2}
\end{aligned} \tag{12}$$

We also find the derivative of Eq. 10 with respect to x with Eq. 13

$$\begin{aligned}
\nabla logP(x) &= \frac{\partial}{\partial \mu_k}\left(log\sum_k \pi_k \frac{1}{\sqrt{2\pi}\sigma_k} exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)\right) \\
&= \frac{1}{\sum_k \pi_k \frac{1}{\sqrt{2\pi}\sigma_k} exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)} \frac{\partial}{\partial \mu_k}\left(\sum_k \pi_k \frac{1}{\sqrt{2\pi}\sigma_k} exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)\right) \\
&= \frac{1}{\sum_k \pi_k \frac{1}{\sqrt{2\pi}\sigma_k} exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)} \left(\sum_k \pi_k \frac{1}{\sqrt{2\pi}\sigma_k} \frac{\partial}{\partial \mu_k} exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)\right) \\
&= \frac{1}{\sum_k \pi_k \frac{1}{\sqrt{2\pi}\sigma_k} exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)} \left(\sum_k \pi_k \frac{1}{\sqrt{2\pi}\sigma_k} exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)\left(\frac{x-\mu_k}{\sigma_k^2}\right)\right) \\
&= \sum_k \frac{\pi_k \frac{1}{\sqrt{2\pi}\sigma_k} exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)}{\sum_k \pi_k \frac{1}{\sqrt{2\pi}\sigma_k} exp\left(\frac{-(x-\mu_k)^2}{2\sigma_k^2}\right)}\left(\frac{x-\mu_k}{\sigma_k^2}\right) \\
&= \sum_k P(z=k|x)\nabla logP(x, z=k)
\end{aligned} \tag{13}$$

Part 5

In order to run gradient descent we need to determine an appropriate cost function, in this case we will be using the marginal data likelihood, or $P(X)$. To implement it we use the results found in the previous section with the steps shown in Eq. 14.

$$\begin{aligned} P(x) &= \prod_n \sum_k P(z_n = k)P(x_n|z_n) \\ &= \prod_n \sum_k P(z_n = k)exp\left(logP(x_n|z_n)\right) \\ &= \prod_n \sum_k exp\left(logP(z_n = k) + logP(x_n|z_n)\right) \\ logP(x) &= \sum_n log\left[\sum_k exp\left(logP(z_n = k) + logP(x_n|z_n)\right)\right] \end{aligned} \tag{14}$$

With the functions previously implemented in Tasks 2 and 3, we can easily code this cost function with

Code 4: Implementation of the Marginal Log Probability.

```
def marginal_log_likelihood(x, logpz, mu, sigma):
    '''
        Calculates the log marginal probability of the data, i.e. logP(x)
    Args:
        x: The data
        pz: Prior probabilities of the clusters
        mu: Cluster means
        sigma: Cluster variance

    Returns:
        Log marginal probability of the data
    '''
    pxn = reduce_logsumexp(tf.transpose(logpz) + log_density(x, mu, sigma),0)
    px = tf.reduce_sum(pxn,0)

    return px
```

We also need reasonable initial values for the parameters and a way to enforce their constraints: $\sum_k P(z_k) = 1$ which we enforce using a softmax function; $\sigma \in [0, \infty)$ which we guarantee with $\sigma^2 = exp(\sigma)$. The code for the initializations and constraints is listed in Code 5.

Code 5: Parameter Initialization and Constraint Enforcement.

```
    pz = tf.Variable(tf.zeros([1,k]))
    logpz = logsoftmax(pz) # Enforce simplex constraint over P(z)

    sigma = tf.Variable(tf.ones([k, 1])*(-3))
    expsigma = tf.exp(sigma) # Enforce sigma > 0

    mu = tf.Variable(tf.random_normal([k, data_d], mean=0, stddev=0.01))
```

With these parameters we train a Mixture of Gaussians on the same dataset used in Parts 4-6 and obtain the results shown in Figure 6 and Table 3 with training curve shown in Figure 7.

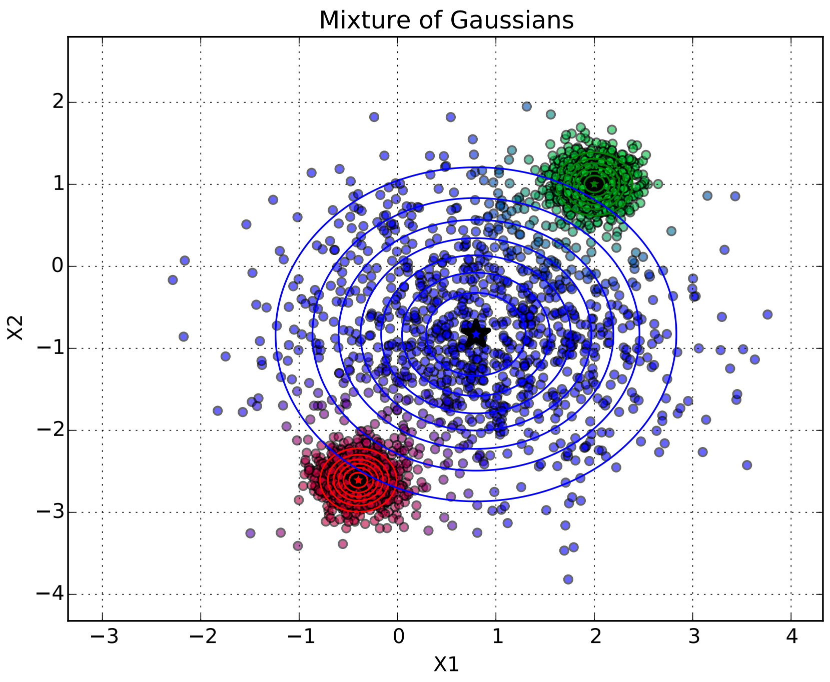

Figure 6: Mixture of Gaussians Model

Cluster	**Probability**	**Mean**	**Standard Deviation**
1	-1.102	[-0.398, -2.607]	0.040
2	-1.098	[2.001, 1.005]	0.038
3	-1.097	[0.797, -0.829]	1.000

Table 3: Evaluation of the model on test set.

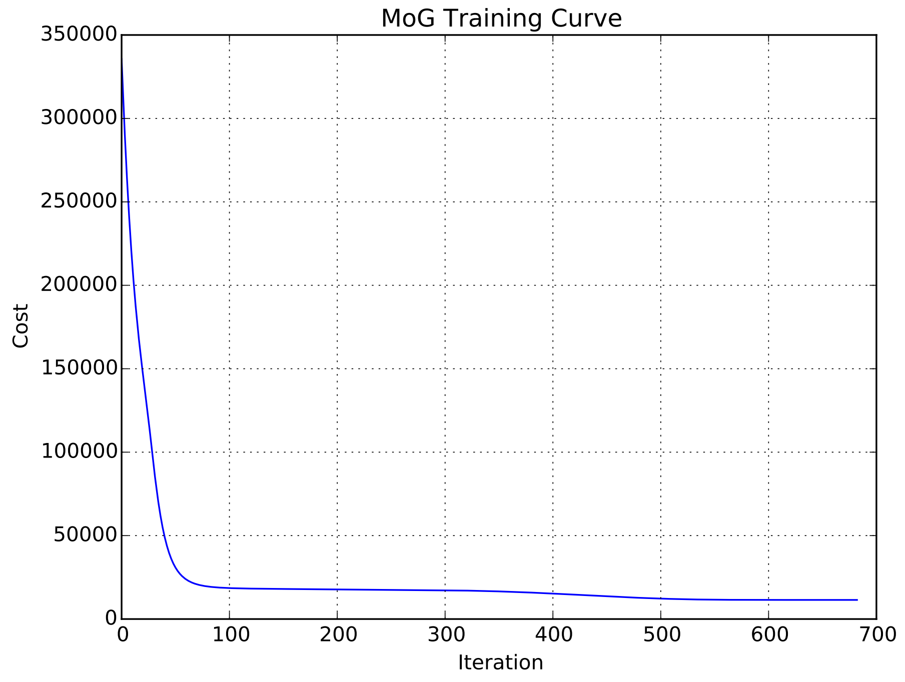

Figure 7: Mixture of Gaussians Model Training Curve

Part 6

In a process similar to the one performed in Parts 5-6, we train Mixture of Gaussian models with the number of clusters varying from 1 to 5. In doing so we obtain the models depicted in Figure 8. The hard assignments inferred from the colors may be misleading as it does not represent the different responsibility levels of each cluster.

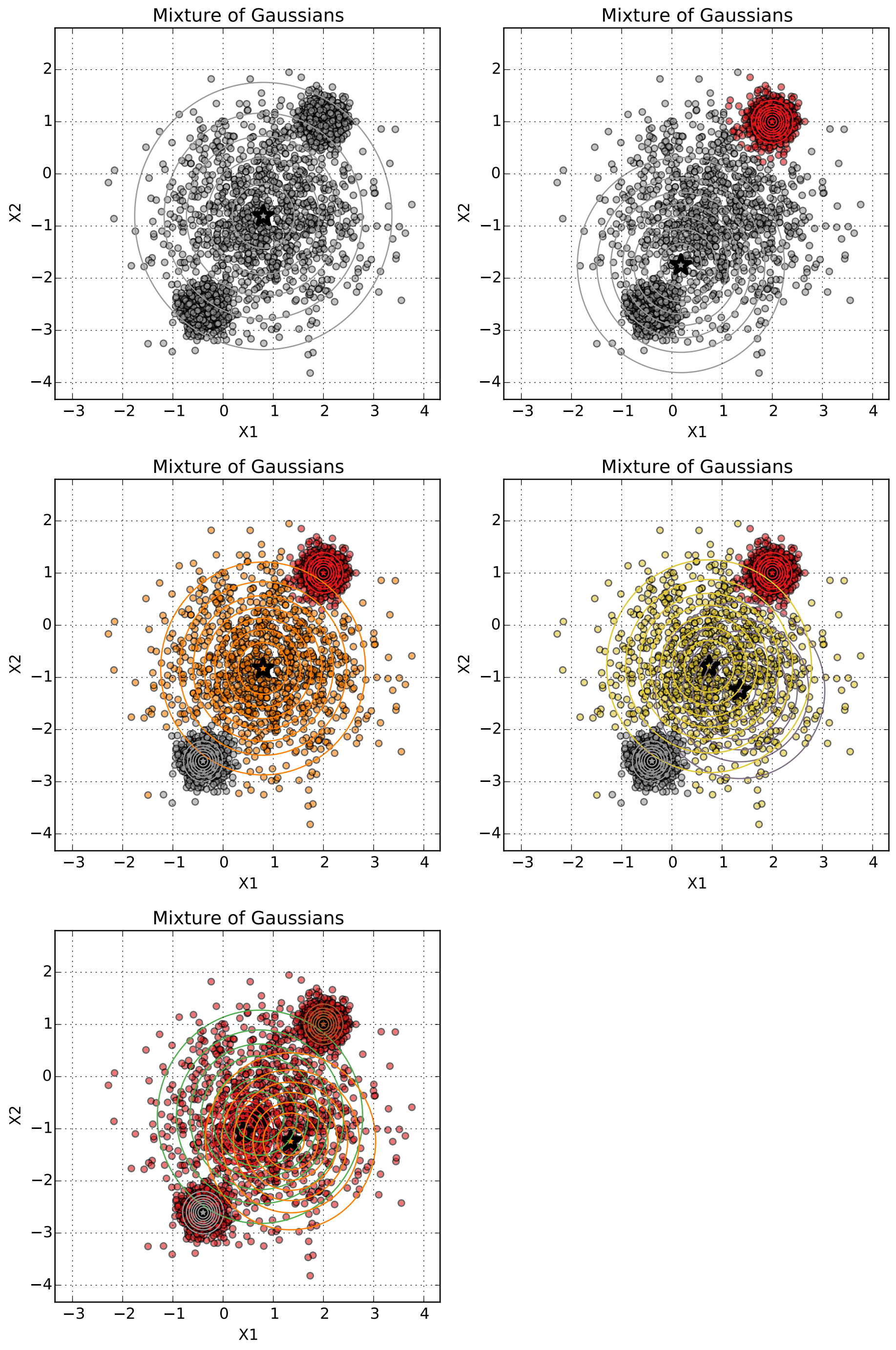

Figure 8: Mixture of Gaussians Models

For each model we obtain the log-likelihood in the test set listed in Table 4 (and also plotted in Figure 9). From the data it is clear that $K = 3$ is the optimal choice of clusters since the log-likelihood plateaus after this value.

Clusters	**Log Likelihood**
1	-11620.993
2	-8054.862
3	-5683.814
4	-5684.218
5	-5683.494

Table 4: Log Likelihood of Each Model on Test Set.

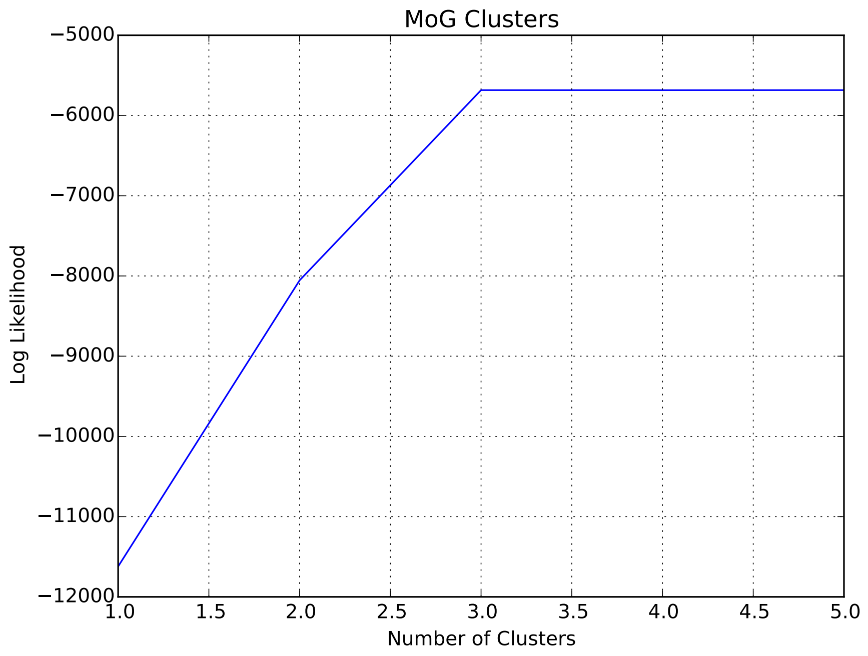

Figure 9: Mixture of Gaussians Models

Part 7

With a 100-dimensional data we can no longer visualize scatter plots to decide the number of clusters, therefore we must rely on the cost functions established. For the Mixture of Gaussians we use log-likelihood and for K-Means we use squared error. Swiping the number of clusters from 1 to 9, we obtain the values listed in Table 5.

Clusters	**MoG Log Likelihood**	**KMeans Squared Error**
1	-22507.699	100.304
2	-22385.707	78.371
3	-22321.289	60.465
4	-22017.779	43.238
5	-21383.684	42.950
6	-21383.762	42.829
7	-21383.740	21.729
8	-21383.795	36.795
9	-21383.828	21.891

Table 5: Log Likelihood of Each Model on Test Set.

Plotting the log-likelihood of the Mixture of Gaussians models we get the picture depicted in Figure 10 from which we can notice that the value plateaus after $K = 5$, so we conclude that this is a reasonable amount of clusters in the dataset.

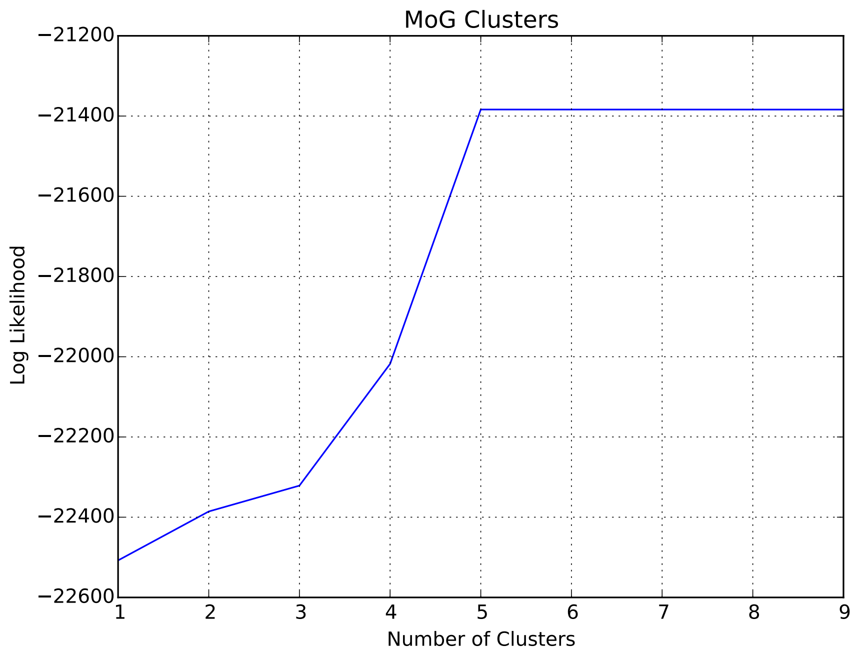

Figure 10: Mixture of Gaussians Models

Now plotting the K-Means error we get Figure 11, from which we notice that K-Means improves very little from $K = 4$ to 6. Therefore to some extent both K-Means and Mixture of Gaussians produce best results with $K = 5$. However, we notice that K-Means does not behave as well with high dimensional data as MoG did or how K-Means did with the 2-dimensional dataset, as can be noticed by the oscillating error values towards the higher number of clusters, suggesting low minima.

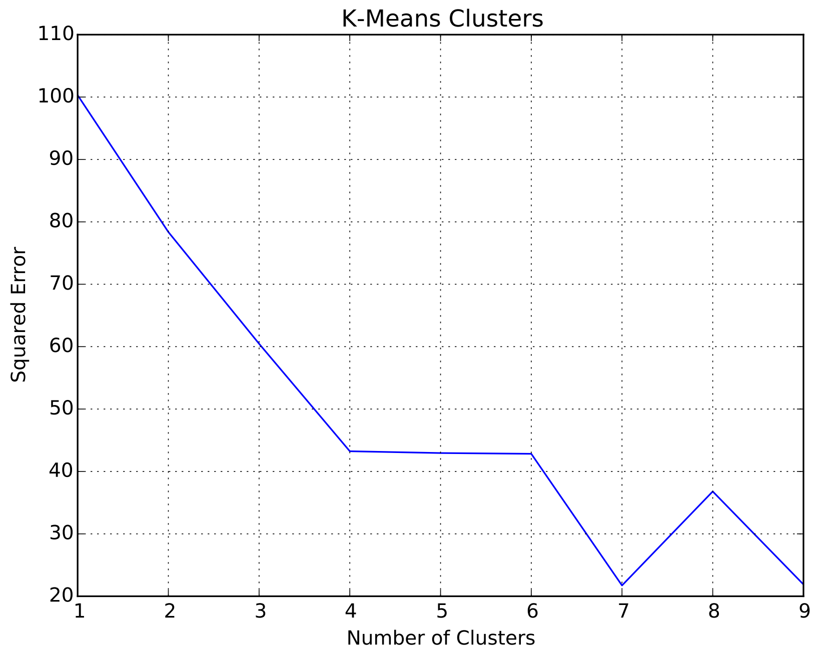

Figure 11: Mixture of Gaussians Models