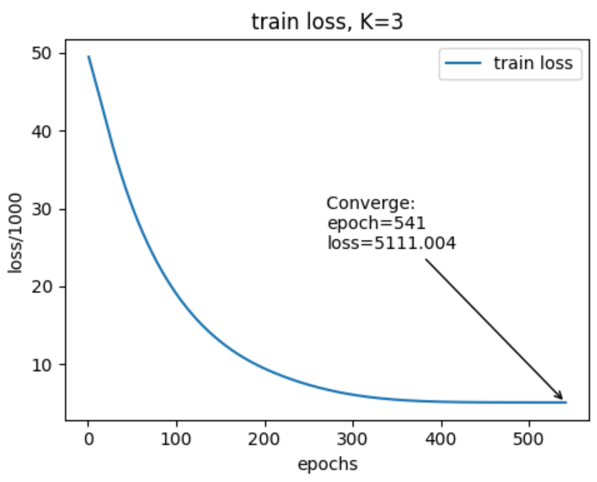
ECE1513 Winter 2019 Assignment 3

April 10, 2019

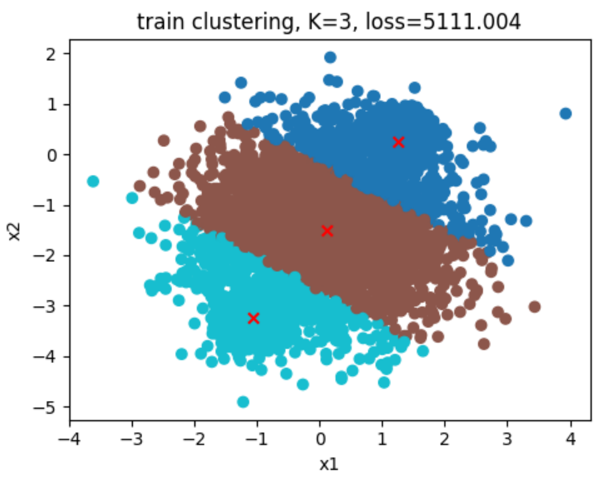
1 K-means

1.1 Learning K-means

1. We implemented K-Means by minimizing loss function $\mathcal{L}(\mu)$ with AdamOptimizer. The training process early stopped at epoch 541 since the loss cannot be further minimizd. The train loss and clustering result are shown in Figure 1.



(a) Training loss



(b) Clustering(K=3)

Figure 1: K-means clustering with K=3

2. We ran the algorithm with $K = 1, 2, 3, 4, 5$ and computed the percentage of the data points belonging to each cluster, as in Table 1. We also plotted the clustering results in Figure 2, to get an intuitive understanding. From the clustering result, we consider $K = 2$ is the best clustering. The reasons are:

- The histogram in Figure 3 shows that the data has two clusters.
- From the percentage data and plots, we can see that for $K = 2$, data points are roughly divided evenly, and the clustering fits the distribution best. Thus $K = 2$ is the best clustering.

- For $K = 3, 4, 5$, the 2D plots do not match the actual clustering and the percentage of each clusters are not distributed evenly. These show that $K = 3, 4, 5$ are not good choices.

K	Cluster %				
	0	1	2	3	4
1	100%				
2	49.5%	50.5%			
3	23.81%	37.97%	38.22%		
4	12.03%	37.31%	37.14%	13.52%	
5	8.57%	36.37%	7.46%	11.71%	35.89%

Table 1: Percentage of the data points belonging to each clusters, K=1,2,3,4,5

.

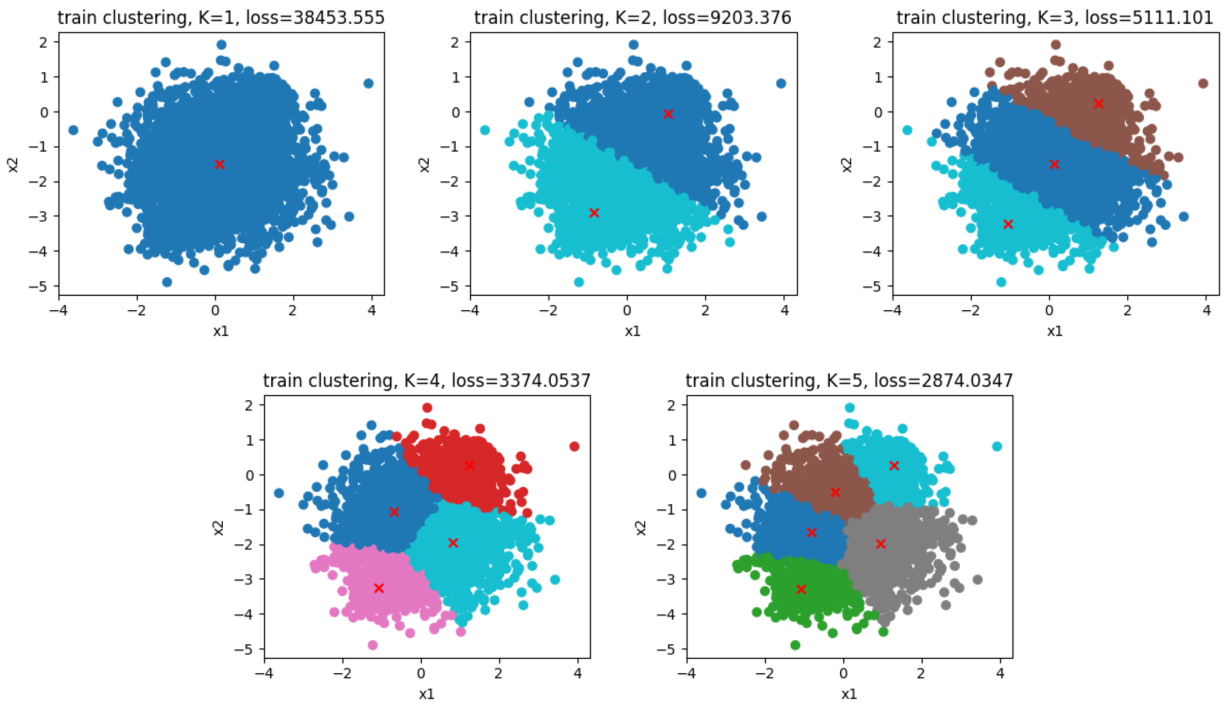


Figure 2: K-means clustering with K=1,2,3,4,5. Red crosses show the positions of clustering centers.

3. We used 1/3 as validation set and trained models on $K = 1, 2, 3, 4, 5$. Since the loss function $\mathcal{L}(\mu) = \sum_{n=1}^{N} \min_{k=1}^{K} \|\mathbf{x}_n - \mu_k\|_2^2$ is related to number of data points, the training loss is expected to be two times greater than validation loss. Thus, we compute the ratio of $2\mathcal{L}_{valid}/\mathcal{L}_{train}$ as the metric of performance. The result is shown in Table 2.

From Table 2 we can see that $K = 2$ has the least valid/train ratio, thus $K = 2$ is the best number of clusters.

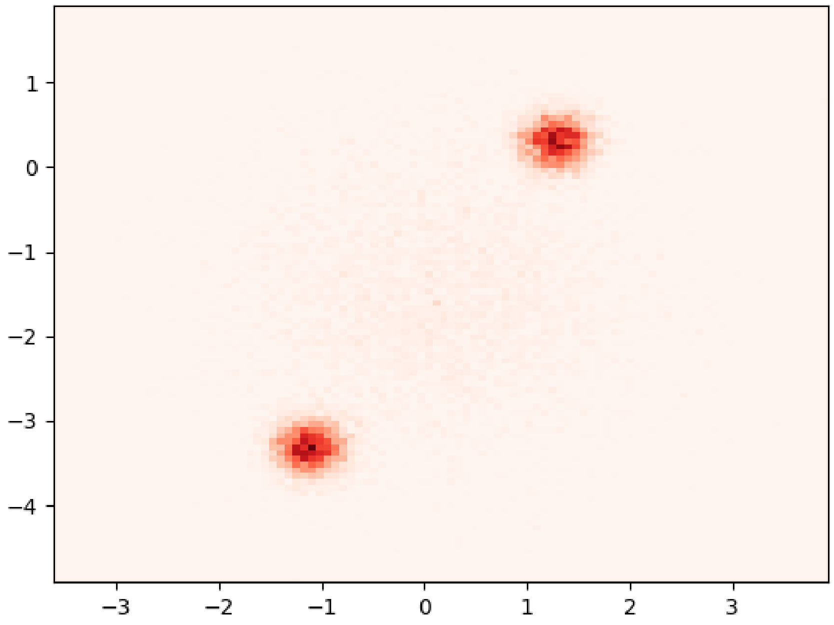


Figure 3: 2D histogram of the training set.

K	$\mathcal{L}_{train}$	$\mathcal{L}_{valid}$	$2 \times \mathcal{L}_{valid}/\mathcal{L}_{train}$
1	25655.162	12800.023	0.99785
2	6231.329	2973.207	0.95428
3	3369.564	1746.5096	1.03664
4	2253.9624	1121.8652	0.99546
5	1866.385	1005.4237	1.07740

Table 2: Training loss vs. Validation loss

2 Mixtures of Gaussians

2.1 The Gaussian cluster mode

1. First we compute the log probability density function for cluster k.

$$\begin{aligned}\log \mathcal{N}(\mathbf{x}; \mu_j, \Sigma_j) &= \log\Big(\frac{1}{(2\pi)^{d/2}\sqrt{|\Sigma|}} \exp\big(-\frac{1}{2}(\mathbf{x}-\mu)^\intercal \Sigma_j^{-1}(\mathbf{x}-\mu)\big)\Big) \\ &= -0.5d\log(2\pi) - 0.5\log|\Sigma| - 0.5(\mathbf{x}-\mu)^\intercal \Sigma_j^{-1}(\mathbf{x}-\mu) \\ &= -0.5\big(d\log(2\pi) + \log|\Sigma| + (\mathbf{x}-\mu)^\intercal \Sigma_j^{-1}(\mathbf{x}-\mu)\big).\end{aligned}$$

Given that for the multivariate Gaussian for cluster k, different data dimensions are *independent* and have the *same standard deviation* σ^k, the covariance for the multivariate

Gaussian is a *diagonal matrix* on $\mathbb{R}^{d\times d}$ with ${\sigma^k}^2$ as its diagonal values. Therefore we have

$$|\Sigma| = ({\sigma^k}^2)^d, \Sigma^{-1} = \mathbf{I}/{\sigma^k}^2.$$

Thus

$$\begin{aligned}\log\mathcal{N}(\mathbf{x};\mu_j,\Sigma_j) &= -0.5\big(d\log(2\pi) + \log({\sigma^k}^{2d}) + (\mathbf{x}-\mu)^\intercal\mathbf{I}(\mathbf{x}-\mu)/{\sigma^k}^2\big)\\ &= -0.5\big(d\log(2\pi) + d\log({\sigma^k}^2) + (\mathbf{x}-\mu)^\intercal(\mathbf{x}-\mu)/{\sigma^k}^2\big).\end{aligned}$$

To enable batch computation in Tensorflow, some dimension manipulations are required. See the code snippet below for details.

```python
def log_pdf(self, X, Mu, Var):
  """
  Compute the log PDF using multivariate Gaussian distribution.
  :param X: (N x D)
  :param Mu: (K x D)
  :param Var: (K,)
  :return (N x K)
  """
  K, D = Mu.shape.as_list()

  # We need to compute (x - mu)^T @ (x - mu) for all k in K, n in N,
  # where (x - mu) is with shape of (D, 1).
  # Thus the actual multiplication should be done with
  # dist_T(N, K, 1, D) @ dist(N, K, D, 1) => (N, K, 1, 1)
  dist = X[:, None, :, None] - Mu[None, :, :, None]      # (N x K x D x 1)
  dist_T = tf.transpose(dist, (0, 1, 3, 2))              # (N x K x 1 x D)

  return -.5 * (
    D * np.log(2 * np.pi) +
    D * tf.log(Var) +
    tf.reshape(tf.matmul(dist_T, dist), shape=(-1, K)) / Var
  )
```

2. First we compute the log probability of the cluster variable $z = k$ given the data vector $\mathbf{x}$:

$$\begin{aligned}P(z=k|\mathbf{x}) &= P(\mathbf{x}, z=k)\Big/\sum_{j=1}^{K} P(\mathbf{x}, z=j)\\ &= P(z=k)P(\mathbf{x}|z=k)\Big/\sum_{j=1}^{K} P(z=j)P(\mathbf{x}|z=j),\end{aligned}$$

where $P(z=j) = \pi_j$ subject to $\pi_j > 0, \sum_{j=1}^{K}\pi_j = 1$, and $P(\mathbf{x}|z=j) = \mathcal{N}(\mathbf{x};\mu_j,{\sigma^j}^2)$.

Thus

$$P(z=k|\mathbf{x}) = \pi_k \mathcal{N}(\mathbf{x};\mu_k,{\sigma^k}^2)\Big/\sum_{j=1}^{K}\pi_j\mathcal{N}(\mathbf{x};\mu_j,{\sigma^j}^2),$$

$$\log P(z=k|\mathbf{x}) = \log\pi_k + \log\mathcal{N}(\mathbf{x};\mu_k,{\sigma^k}^2) - \log\sum_{j=1}^{K}\pi_j\mathcal{N}(\mathbf{x};\mu_j,{\sigma^j}^2).$$

Denote $\pi_k = e^{\psi_k}/\sum_k e^{\psi_k}, {\sigma^j}^2 = e^{\phi_k}$, then

$$\begin{aligned}
\log\mathcal{N}(\mathbf{x};\mu_k,{\sigma^k}^2) &= \mathrm{logpdf}(\mathbf{x};\mu_k,e^{\phi_k}),\\
\log\pi_k &= \log e^{\psi_k} - \log\sum_k e^{\psi_k} = \psi_k - \mathrm{logsumexp}(\psi),\\
\log\pi &= [\log\pi_1\cdots\log\pi_K]\\
&= [\psi_1 - \mathrm{logsumexp}(\psi)\cdots\psi_K - \mathrm{logsumexp}(\psi)]\\
&= [\psi_1\cdots\psi_K] - \mathrm{logsumexp}(\psi)\\
&= \psi - \mathrm{logsumexp}(\psi) = \mathrm{logsoftmax}(\psi).
\end{aligned}$$

Thus

$$\begin{aligned}
\log P(z|\mathbf{x}) &= [\log(\pi_1) + \mathrm{logpdf}(\mathbf{x},\mu_1,e^{\phi_1}) - \mathrm{logsumexp}(\log\pi + \mathrm{logpdf}(\mathbf{x},\mu,e^{\phi}))\cdots]\\
&= \log\pi + \mathrm{logpdf}(\mathbf{x},\mu,e^{\phi}) - \mathrm{logsumexp}(\log\pi + \mathrm{logpdf}(\mathbf{x},\mu,e^{\phi}))\\
&= \mathrm{logsoftmax}(\psi) + \mathrm{logpdf}(\mathbf{x},\mu,e^{\phi}) - \mathrm{logsumexp}(\mathrm{logsoftmax}(\psi) + \mathrm{logpdf}(\mathbf{x},\mu,e^{\phi}))
\end{aligned}$$

You may note that above equation can be further reduced as $\mathrm{logsoftmax}(\mathrm{logsoftmax}(\psi) + \mathrm{logpdf}(\mathbf{x},\mu,e^{\phi}))$. However, when considering multiple data points $\mathbf{X}\in\mathbb{R}^{N\times D}$, the result of $\mathrm{logsoftmax}(\psi) + \mathrm{logpdf}(\mathbf{x},\mu,e^{\phi})\in\mathbb{R}^{N\times K}$, thus for each single data point, we need the softmax to be computed along the 2nd axis (i.e. the axis with size K), not the 1st axis (the axis with size N). Since the helper function `logsoftmax` only computes along the first axis, we cannot use `logsoftmax` directly here.

```python
def log_proba(self, X, Mu, Phi, Psi):
  """Compute the log probability of the cluster variable z given the data vector x.
  :param X: data matrix, (N x D)
  :param Mu: cluster center matrix, (K x D)
  :param Phi: (K,)
  :param Psi: (K,)
  :return (N x K)
  """
  N, D = X.shape
  K = Mu.shape[0]

  logpdf = self.log_pdf(X, Mu, tf.exp(Phi))      # (N x K)
  logw = logsoftmax(Psi)            # (K,)
```

```
logw_pdf = logpdf + logw          # (N x K)
lse = reduce_logsumexp(logw_pdf, reduction_indices=1, keep_dims=True)   # (N, 1)

return logw_pdf - lse             # (N x K)
```

The reason for using `logsumexp` instead of `tf.reduce_sum` is that, by subtracting the maximum value from all the components, `logsumexp` can prevent data overflow when the vector is extremely large.

2.2 Learning the MoG

1. First we compute the equation of loss function, with $\pi_k = e^{\psi_k} / \sum_k e^{\psi_k}$, ${\sigma^j}^2 = e^{\phi_k}$:

$$\begin{aligned}
\mathcal{L}(\mu, \psi, \phi) &= -\log \prod_n \sum_k \pi^k \mathcal{N}(\mathbf{x}_n; \mu^k, {\sigma^k}^2) \\
&= -\sum_n \log \sum_k \pi^k \mathcal{N}(\mathbf{x}_n; \mu^k, {\sigma^k}^2) \\
&= -\sum_n \log \sum_k \exp\left(\log \pi^k + \log \mathcal{N}(\mathbf{x}_n; \mu^k, {\sigma^k}^2)\right) \\
&= -\sum_n \text{logsumexp}\Big(\log \pi + \log \mathcal{N}(\mathbf{x}_n; \mu, \sigma^2)\Big) \\
&= -\sum_n \text{logsumexp}\Big(\text{logsoftmax}(\psi) + \text{logpdf}(\mathbf{x}_n, \mu, e^{\phi})\Big).
\end{aligned}$$

We implemented the loss function and trained a MoG model with $K = 3$. After early stop at epoch 511, the best parameters are:

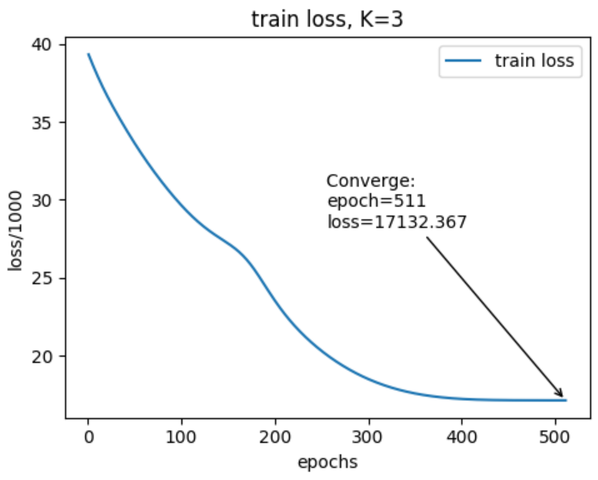
$$\mu = \begin{bmatrix} -1.1011823 & -3.3061764 \\ 1.298578 & 0.3091123 \\ 0.10558542 & -1.5281174 \end{bmatrix}, \phi = \begin{bmatrix} -3.242426 \\ -3.2409697 \\ -0.01329159 \end{bmatrix}, \psi = \begin{bmatrix} 0.13701023 \\ 0.14201044 \\ 0.1449725 \end{bmatrix}$$

The loss plot and clustering are shown in Figure 4. For demonstration purpose we also plot the contour of each Gaussian.

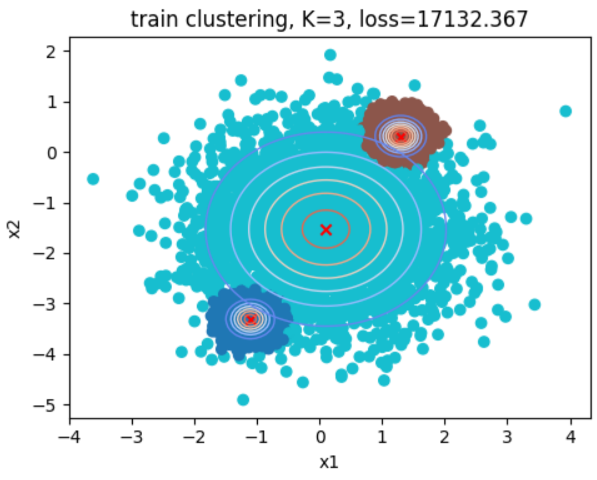
2. Table 3 shows the training and validation loss for $K = 1, 2, 3, 4, 5$. Since the loss function is related with data size, we compute the ratio of validation loss and training loss. The minimum of ratio appears at $K = 3$, indicating that $K = 3$ is the best clustering.

The clustering plot of each K value is shown in Figure 5.

3. We run both K-means and MoG on dataset *data100D.npy* and plot the clustering in Figure 7 and Figure 8. The 100-D input data is reduced to 2-D with PCA, while PC1 and PC2 account for 24% and 21% of the variation respectively. Intuitively $K = 10$ is the best clustering for both algorithms, however we need a quantitative metric to evaluate their performance. We choose *silhouette score* as the metric of clustering. *Silhouette score* is a metric of how similar a data point is similar to its own cluster than to other clusters. Given



(a) Training loss



(b) Clustering(K=3)

Figure 4: Mixture of Gaussians clustering with K=3. Red cross shows the clustering center while the contour shows the density of each Gaussian distribution.

K	$\mathcal{L}_{train}$	$\mathcal{L}_{valid}$	$2\mathcal{L}_{valid}/\mathcal{L}_{train}$
1	23296.97	11619.322	0.99750
2	16016.173	8125.6406	1.01468
3	11458.64	5678.3496	0.99110
4	11354.237	5782.4033	1.01855
5	11317.508	5819.642	1.02843

Table 3: Training loss vs. Validation loss

point $\mathbf{x}$, denote the mean intra-cluster distance as a and the mean nearest-cluster distance as b, then the silhouette score is $(b-a)/\max(a,b)$.

Both K-means and MoG cannot guarantee global minima when using gradient descent as optimization. Thus we run the clustering 10 times for each algorithm and compute the average silhouette score with `sklearn.metric.silhouette_score`. The result score is shown in Table 4 and Figure 6.

The silhouette scores show that for K-means and MoG, the best clusterings are $K = 10$ and $K = 15$ respectively.

k	K-means mean	K-means stddev	MoG mean	MoG stddev
5	0.517469	0.043120	0.380841	0.088725
10	$\underline{0.528677}$	0.100117	0.530042	0.051024
15	0.489333	0.083503	$\underline{0.552551}$	0.083270
20	0.451246	0.093502	0.536086	0.065657
30	0.402536	0.061891	0.469634	0.077099

Table 4: Mean and standard deviation of silhouette scores for 10 runs of K-means and MoG. The best clusterings are underlined.

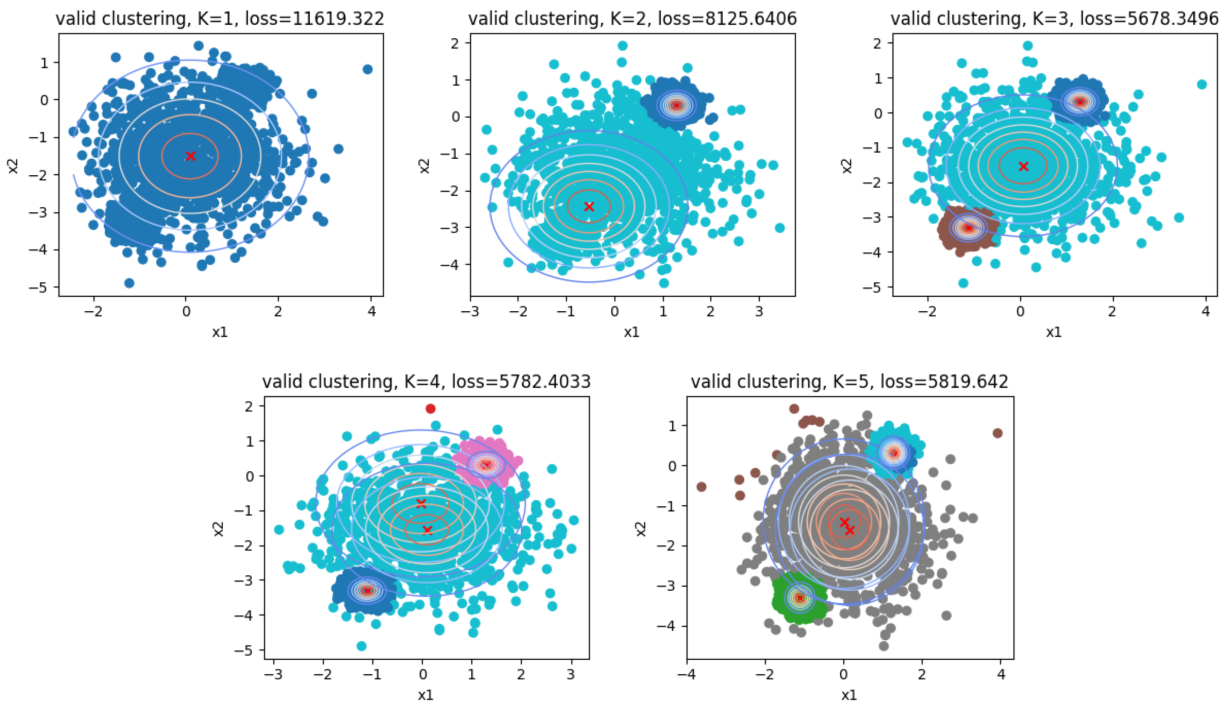


Figure 5: Mixture of Gaussians clustering with K=1,2,3,4,5

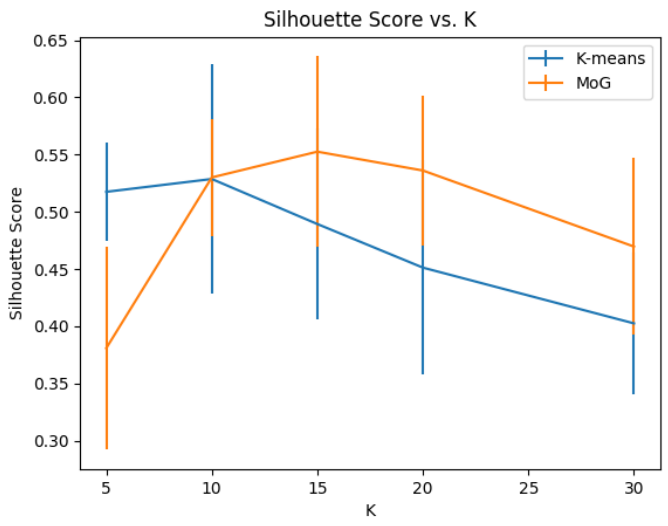


Figure 6: Silhouette score for K=5,10,15,20,30

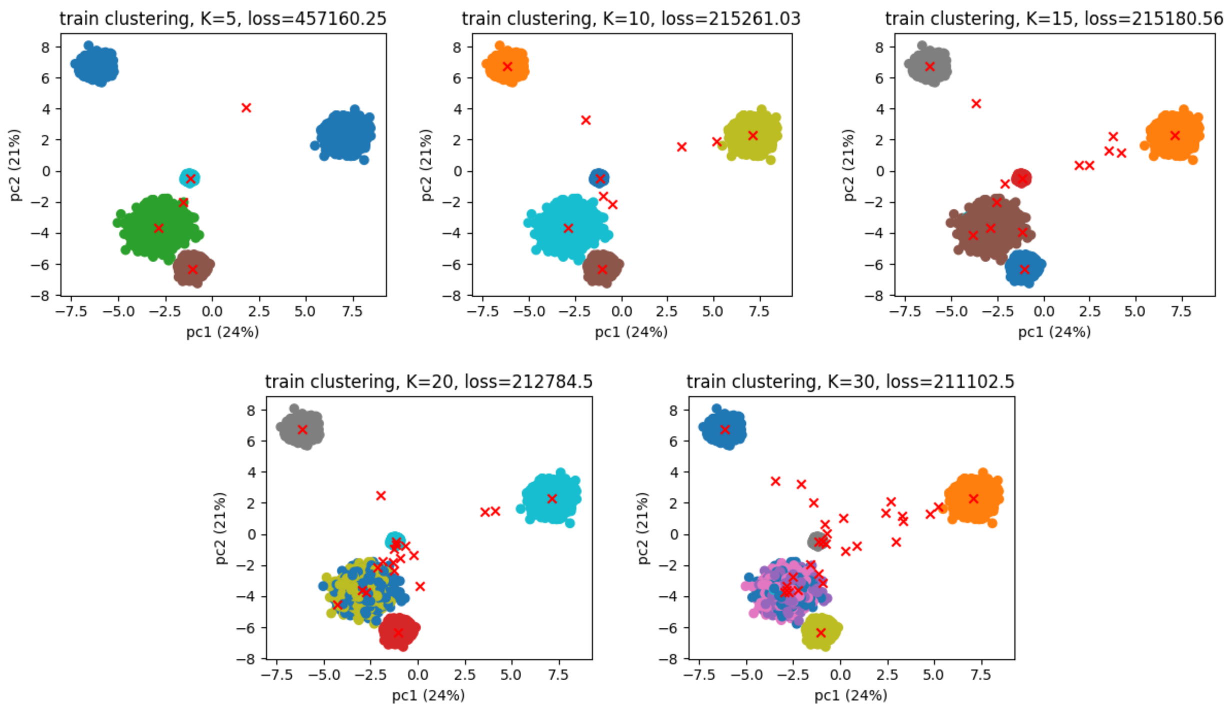


Figure 7: K-means clustering with K=1,2,3,4,5 on data100D

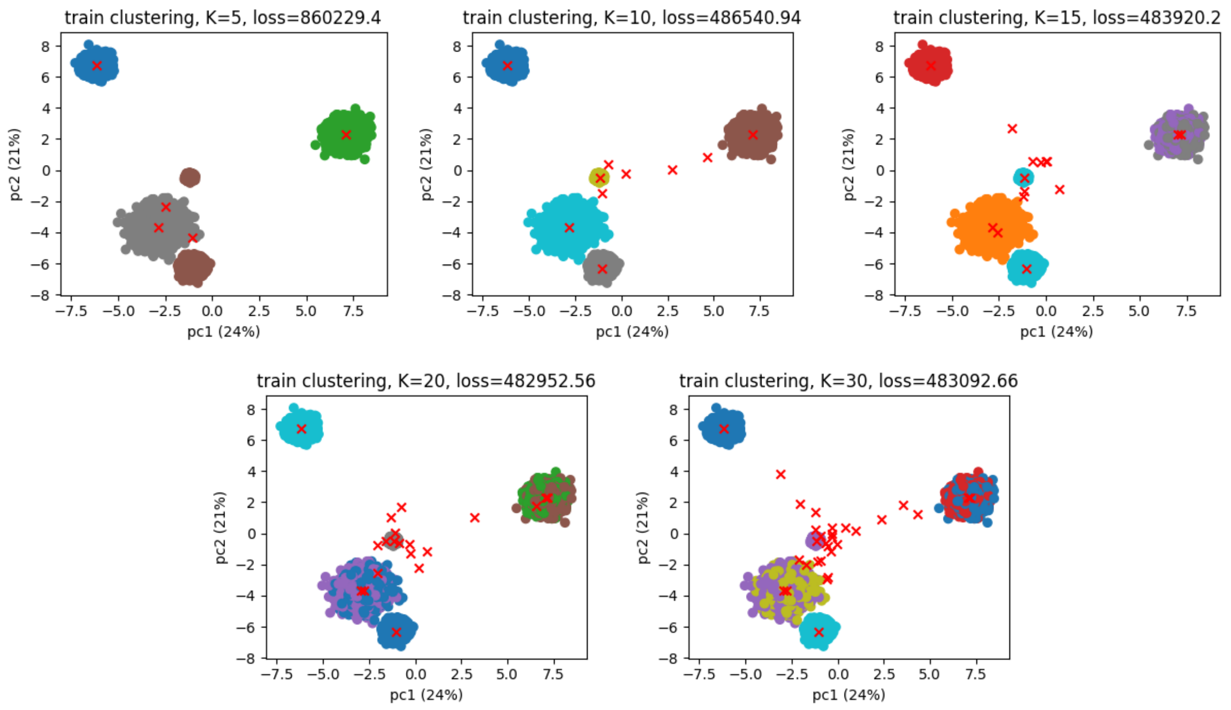


Figure 8: MoG clustering with K=1,2,3,4,5 on data100D