

# Dirichlet Process Mixture Models

## General Principles

To discover group structures or clusters in data without pre-specifying the number of groups, we can use a **Dirichlet Process Mixture Model (DPMM)** Gershman and Blei (2012). This is a unsupervised clustering method . Essentially, the model assumes the data is generated from a collection of different Gaussian distributions, and it simultaneously tries to figure out:

1. **How many clusters ( $K$ ) exist:** Unlike algorithms like K-Means, the DPMM infers the most probable number of clusters directly from the data.
2. **The properties of each cluster:** For each inferred cluster, it estimates its location and its spread.
3. **The assignment of each data point:** It determines the probability of each data point belonging to each cluster.

## Considerations

### 🔥 Caution

- A DPMM is a Bayesian model that considers uncertainty in all its parameters. The core idea is to use the Dirichlet Process prior that allows for a potentially infinite number of clusters. In practice, we use a finite approximation where we cap the maximum number of clusters at  $K$  and use the Stick-Breaking Process .
  - The key parameters and their priors are:
    - **Concentration  $\alpha$ :** This single parameter controls the tendency to create new clusters. A low favors fewer, larger clusters, while a high allows for many smaller clusters. We typically place a **Gamma** prior on  $\alpha$  to learn its value from the data.
    - **Cluster Weights  $w$ :** Generated via the Stick-Breaking process from  $\alpha$ . These are the probabilities of drawing a data point from any given cluster.

- **Cluster Parameters**  $(\mu, \Sigma)$ : Each potential cluster has a mean  $\mu$  and a covariance matrix  $\Sigma$ . If the data have multiple dimensions, we use a multivariate normal distribution (see chapter, 14). However, if the data is one-dimensional, we use a univariate normal distribution.
- The model is often implemented in its marginalized form . Instead of explicitly assigning each data point to a cluster, we integrate out this choice. This creates a smoother probability surface for the inference algorithm to explore, leading to much more efficient computation.

## Example

Below is an example of a DPMM implemented in BI. The goal is to cluster a synthetic dataset into its underlying groups. The code first generates data with 4 distinct centers and then applies the DPMM to recover these clusters.

## Python

```
from BI import bi, jnp
from sklearn.datasets import make_blobs
import numpyro

m = bi(rand_seed = False)

# Generate synthetic data
data, true_labels = make_blobs(
    n_samples=500, centers=8, cluster_std=0.8,
    center_box=(-10,10), random_state=101
)
data_mean = jnp.mean(data, axis=0)
data_std = jnp.std(data, axis=0)*2

# The model
def dpmm(data, K, data_mean, data_std):
    N, D = data.shape # Number of features

    # 1) stick-breaking weights
    alpha = m.dist.gamma(1.0, 10.0, name='alpha')
```

```

with m.dist.plate("beta_plate", K - 1):
    beta = m.dist.beta(1, alpha, name = "beta")

w = numpyro.deterministic("w",m.models.dpmm.mix_weights(beta))

# 2) component parameters
with m.dist.plate("components", K):
    mu = m.dist.multivariate_normal(loc=data_mean, covariance_matrix=data_std*jnp.eye(D))
    sigma = m.dist.log_normal(0.0, 1.0, shape=(D,), event=1, name='sigma')# shape (T, D)
    Lcorr = m.dist.lkj_cholesky(dimension=D, concentration=1.0, name='Lcorr')# shape (T, D)

    scale_tril = sigma[..., None] * Lcorr # shape (T, D, D)

# 3) Latent cluster assignments for each data point
m.dist.mixture_same_family(
    mixing_distribution=m.dist.categorical(probs=w, create_obj=True),
    component_distribution=m.dist.multivariate_normal(
        loc=mu,
        scale_tril=scale_tril,
        create_obj=True
    ),
    obs=data
)

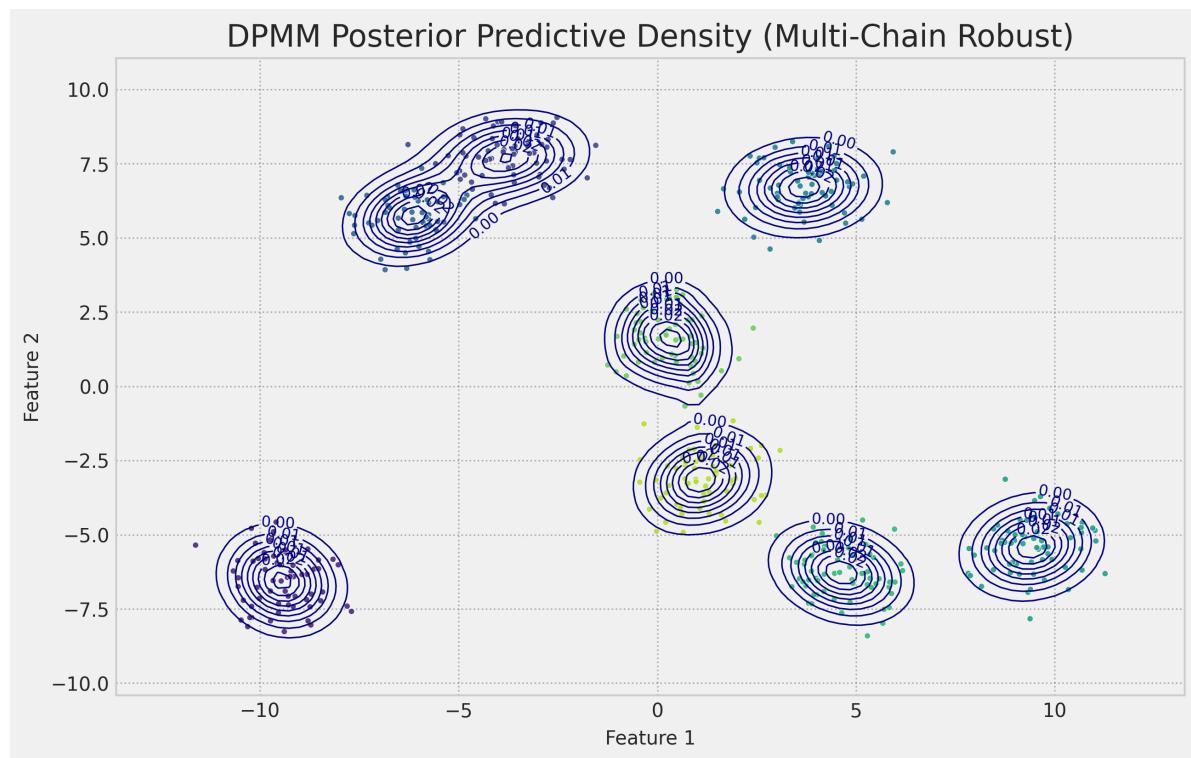
m.data_on_model = dict(data=data, K = 10, data_mean=data_mean, data_std=data_std)
m.fit(dpmm) # Optimize model parameters through MCMC sampling
m.plot(X=data, sampler=m.sampler) # Prebuild plot function for GMM

```

jax.local\_device\_count 16

0%| 0/1000 [00:00<?, ?it/s] warmup: 0%| 1/1000 [00:04<1:07:22, 4.00it/s]

Model found 8 clusters.



R



## Julia

```
using BayesianInference
using PythonCall
numpyro = pyimport("numpyro")

m = importBI(rand_seed = false)

# 1. Generate Data
sk_datasets = pyimport("sklearn.datasets")
output = sk_datasets.make_blobs(n_samples=500, centers=8, cluster_std=0.8, center_box=(-10, 10))
data = output[0]
```

```

data_mean = jnp.mean(data, axis=0)
data_std = jnp.std(data, axis=0) * 2
m.data_on_model = pydict(data=data, K=10, data_mean = data_mean, data_std = data_std)

@OBI function dpmm(data, K, data_mean , data_std)
    N, D = data.shape

    alpha = m.dist.gamma(1.0, 10.0, name="alpha")

    beta = pywith(m.dist.plate("beta_plate", K - 1)) do _
        m.dist.beta(1, alpha, name = "beta")
    end

    w = numpyro.deterministic("w", m.models.dpmm.mix_weights(beta))

    mu, scale_tril = pywith(m.dist.plate("components", K)) do _
        mu_val = m.dist.multivariate_normal(
            loc=data_mean,
            covariance_matrix=data_std * jnp.eye(D),
            name="mu"
        )

        sigma = m.dist.log_normal(0.0, 1.0, shape=(D,), event=1, name="sigma")
        Lcorr = m.dist.lkj_cholesky(dimension=D, concentration=1.0, name="Lcorr")
        scale_tril_inner = jnp.expand_dims(sigma, -1) * Lcorr
        (mu_val, scale_tril_inner)
    end

    m.dist.mixture_same_family(
        mixing_distribution=m.dist.categorical(probs=w, create_obj=true),
        component_distribution=m.dist.multivariate_normal(
            loc=mu,
            scale_tril=scale_tril,
            create_obj=true
        ),
        obs=data
    )
end

# 4. Run

```

```
m.fit(dpmm)

@pyplot m.models.dpmm.plot_dpmm(m.data_on_model["data"], m.sampler)
```

## Mathematical Details

The process involves two keys submodels. The first, aims to identify the location and scale of  $K$  potential clusters. The second, aims to identify which cluster is most likely to have generated a given data point.

$$\begin{pmatrix} Y_{i,1} \\ \vdots \\ Y_{i,D} \end{pmatrix} \sim \text{MVN} \left( \begin{pmatrix} \mu_{z_i,1} \\ \vdots \\ \mu_{z_i,D} \end{pmatrix}, \Sigma_{z_i} \right)$$

$$\begin{pmatrix} \mu_{k,1} \\ \vdots \\ \mu_{k,D} \end{pmatrix} \sim \text{MVN} \left( \begin{pmatrix} A_1 \\ \vdots \\ A_D \end{pmatrix}, B \right)$$

$$\Sigma_k = \text{Diag}(\sigma_k) \Omega_k \text{Diag}(\sigma_k)$$

$$\sigma_{[k,d]} \sim \text{HalfCauchy}(1)$$

$$\Omega_k \sim \text{LKJ}(2)$$

$$z_i \sim \text{Categorical}(\pi)$$

$$\pi_i(\beta_{1:K}) = \beta_i \prod_{j < K} (1 - \beta_j)$$

$$\beta_k \sim \text{Beta}(1, \alpha)$$

$$\alpha \sim \text{Gamma}(1, 10)$$

Where :

- $\begin{pmatrix} Y_{[i,1]} \\ \vdots \\ Y_{[i,D]} \end{pmatrix}$  is the  $i$ -th observation of a D-dimensional data array.

- $\begin{pmatrix} \mu_{[k,1]} \\ \vdots \\ \mu_{[k,D]} \end{pmatrix}$  is the  $k$ -th parameter vector of dimension D.
- $\begin{pmatrix} A_1 \\ \vdots \\ A_D \end{pmatrix}$  is a prior for the mean vector as derived from mean of the raw data.
- $B$  is the prior covariance of the cluster means, and is setup as a diagonal matrix with 0.1 along the diagonal.
- $\Sigma_k$  is the DxD covariance matrix of the  $k$ -th cluster (it is composed from  $\sigma_k$  and  $\Omega_k$ ).
- $\text{Diag}(\sigma_k)$  is a diagonal matrix whose diagonal entries are the standard deviations:

$$\text{Diag}(\sigma_k) = \begin{pmatrix} \sigma_{[k,1]} & 0 & \cdots & 0 \\ 0 & \sigma_{[k,2]} & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & \sigma_{[k,D]} \end{pmatrix}.$$

- $\sigma_k$  is a  $D$ -vector of standard deviations for the  $k$ -th cluster where each element,  $d$ , has a half-cauchy prior.
- $\Omega_k$  is a correlation matrix for the  $k$ -th cluster.
- $z_i$  is a latent variable that maps observation  $i$  to cluster  $k$ .
- $\pi$  is a vector of  $K$  cluster weights, some of which may be close to zero if the predicted number of clusters is less than the maximum number of clusters.
- $\beta_k$ : The set of  $K$  Beta-distributed random variables used in the stick-breaking process to construct the mixture weights.
- $\alpha$ : The concentration parameter, controlling the effective number of clusters.

## Notes

### Note

- The primary advantage of the DPMM is the **automatic inference of the number of clusters**. The posterior distribution of the weights  $w$  reveals which components are “active”, giving a probabilistic estimate of  $K$ .
- Prior  $\alpha$  strongly influence the predicted number of clusters. Below are examples of this relationship:

Table 1: Impact of Gamma Prior Hyperparameters on Cluster Counts

Shape	Rate	Behavior
1	15	Forces very few clusters
5	1	Encourages many small clusters
10	2	Same mean, less variance
2	0.5	Moderately many clusters
15	1	Explosive prior cluster count

## Reference(s)

[https://en.wikipedia.org/wiki/Dirichlet\\_process](https://en.wikipedia.org/wiki/Dirichlet_process) [https://pyro.ai/examples/dirichlet\\_process\\_mixture.html](https://pyro.ai/examples/dirichlet_process_mixture.html)

Gershman, Samuel J, and David M Blei. 2012. “A Tutorial on Bayesian Nonparametric Models.” *Journal of Mathematical Psychology* 56 (1): 1–12.