

Gaussian Mixture Models

General Principles

To discover group structures or clusters in data, we can use a **Gaussian Mixture Model (GMM)**. This is a parametric clustering method. A GMM assumes that the data is generated from a mixture of a **pre-specified number (K)** of different Gaussian distributions. The model's goal is to figure out:

1. **The properties of each of the K clusters:** For each of the K clusters, it estimates its center (mean μ) and its shape/spread (covariance Σ).
2. **The mixture weights:** It estimates the proportion of the data that belongs to each cluster.
3. **The assignment of each data point:** It determines the probability of each data point belonging to each of the K clusters.

Considerations

🔥 Caution

- A GMM is a Bayesian model that considers uncertainty in all its parameters, *except for the number of clusters, K* , which must be fixed in advance.
- The key parameters and their priors are:
 - **Number of Clusters K :** This is a **fixed hyperparameter** that you must choose before running the model. Choosing the right K often involves running the model multiple times and using model comparison criteria (like cross-validation, AIC, or BIC).
 - **Cluster Weights w :** These are the probabilities of drawing a data point from any given cluster. Since there are a fixed number K of them and they must sum to 1, they are typically given a **Dirichlet prior**. A symmetric **Dirichlet prior** (e.g., `Dirichlet(1, 1, ..., 1)`) represents an initial belief that all clusters are equally likely.

- **Cluster Parameters (μ , Σ): Each of the K clusters has a mean μ and a covariance matrix Σ . We place priors on these to define our beliefs about their plausible values.
- Like the DPMM, 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.
- To increase accuracy we run a k-means algorithm to initialize the cluster mean priors.

Example

Below is an example of a GMM implemented in BI. The goal is to cluster a synthetic dataset into a pre-specified $K=4$ groups.

Python

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

m = bi()
# Generate synthetic data
data, true_labels = make_blobs(
    n_samples=500, centers=8, cluster_std=0.8,
    center_box=(-10,10), random_state=101
)

# The model
def gmm(data, K, initial_means): # Here K is the *exact* number of clusters
    D = data.shape[1] # Number of features
    alpha_prior = 0.5 * jnp.ones(K)
    w = m.dist.dirichlet(concentration=alpha_prior, name='weights')

    with m.dist.plate("components", K): # Use fixed K
        mu = m.dist.multivariate_normal(loc=initial_means, covariance_matrix=0.1*jnp.eye(D),
            sigma = m.dist.half_cauchy(1, shape=(D,), event=1, name='sigma')
            Lcorr = m.dist.lkj_cholesky(dimension=D, concentration=1.0, name='Lcorr')
```

```

scale_tril = sigma[..., None] * Lcorr

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, crea
    name="obs",
    obs=data
)

# Kmeans clustering do initiate the means
m.ml.KMEANS(data, n_clusters=8)
m.data_on_model = {"data": data,"K": 8 }
m.data_on_model['initial_means'] = m.ml.results['centroids']

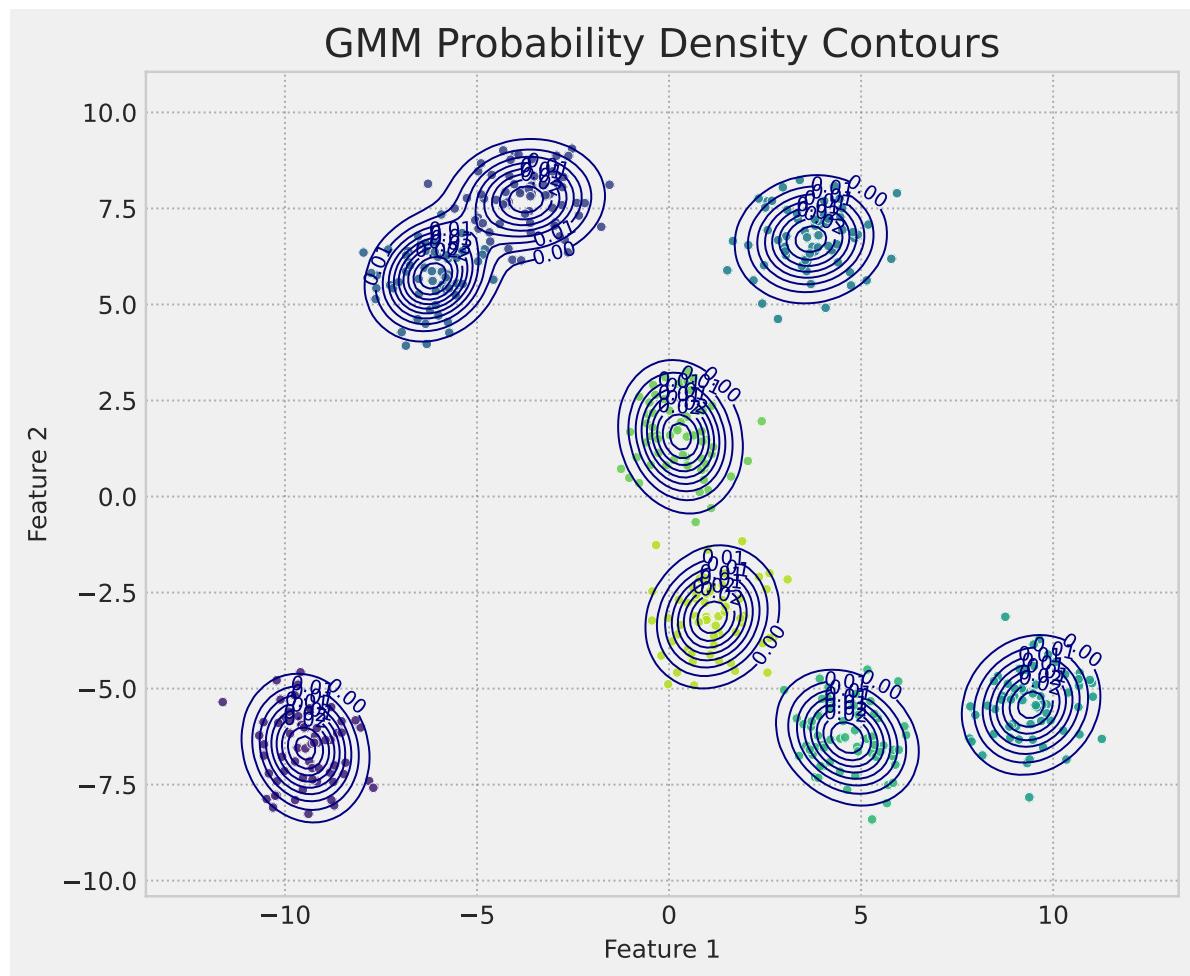
m.fit(gmm) # Optimize model parameters through MCMC sampling
m.plot(X=data,sampler=m.sampler) # Prebuild plot function for GMM

```

jax.local_device_count 32

0%| 0/1000 [00:00<?, ?it/s] warmup: 0%| 1/1000 [00:00<16:10, 1.03it/s]

This function is still in development. Use it with caution.
This function is still in development. Use it with caution.



R



Julia

```
©BI function gmm(data, K, initial_means)
D = data.shape[1]
alpha_prior = 0.5 * jnp.ones(K)
w = m.dist.dirichlet(concentration=alpha_prior, name="weights")

# We capture the output of the pywith block
# The block returns a tuple (mu, scale_tril)
mu, scale_tril = pywith(m.dist.plate("components", K)) do _
```