Unsupervised BNNC

General Principles

Building upon the Dirichlet Process Mixture Model (DPMM) and the Multiclass classification BNN, the **Bayesian Neural Network Clustering (BNNC)** model can learn to separate unlabelled datasets into K underlying mixture components using a deep gating network without observing any initial ground-truth classes.

Rather than sampling discrete cluster assignments via categorical weights directly as a standard GMM, BNNC routes data points through a neural layer to output continuous mixing probabilities θ_i.

Considerations

> **Note**
>
> - **Marginalized Likelihood**: The likelihood of any given observation is constructed by summing the probability density over all K potential multivariate Gaussian clusters recursively weighted by the network's generated θ_i mixing assignments (LogSumExp trick).
> - **Parsimony Prior**: We enforce an explicit Dirichlet prior $Dir(\alpha/K)$ over the gating network weights. Supplying a concentration parameter $\alpha < 1$ acts as a severe sparsity constraint, ensuring the network drives unused clusters towards probability 0, mirroring the stick-breaking properties of DPMMs automatically.
> - **Consensus Clustering**: To determine actual hard cluster sizes after MCMC sampling, we construct an average adjacency matrix measuring how often points are paired together, and run hierarchical clustering on the complement distance matrix.

Example

Below is an example code snippet demonstrating *Unsupervised BNNC* using the BayesForge (**BF**) package to locate hidden clusters from a $K = 4$ synthetic target.

Simulated Data

Python (Raw)

```python
from BayesForge import bf
import jax.numpy as jnp
import jax
import numpyro
import jax.scipy.stats as stats
from sklearn.datasets import make_blobs

# Setup device-------------------------------------------------
m = bf(platform='cpu')

# Generate Synthetic Data -------------------------------------
# 4 distinct underlying true clusters
data, true_labels = make_blobs(
    n_samples=500, centers=4, cluster_std=0.8,
    center_box=(-10,10), random_state=101
)
m.data_on_model = dict(data=data)

# Define raw model --------------------------------------------------
def bnn_mixture_model(data, K=11, D_H1=10):
    # data shape: (N, D)
    N, D_X = data.shape

    # --- BNN Gating Network ---
    w1 = m.bnn.layer_linear(
        data,
        dist=m.dist.normal(0, 1, name='w1_weight', shape=(D_X, D_H1)),
        activation='tanh'
    )

    w2 = m.bnn.layer_linear(
        w1,
        dist=m.dist.normal(0, 0.05, name='w2_weight', shape=(D_H1, K))
    )

    # Global parsimony prior: Dirichlet prior on global cluster weights
    # Alpha < 1 acts as a sparse prior to suppress inactive clusters
    alpha = 0.05
```

```
    pi = numpyro.sample('global_pi', numpyro.distributions.Dirichlet(jnp.ones(K) * (alpha / K

    logits = w2 + jnp.log(pi + 1e-10)
    log_p = jax.nn.log_softmax(logits, axis=-1)
    theta = jnp.exp(log_p)
    numpyro.deterministic('theta', theta)

    # --- Mixture Components ---
    mu = m.dist.normal(0, 5, name='mu', shape=(K, D_X))
    sigma = m.dist.exponential(1.0, name='sigma', shape=(K, D_X))

    # --- Marginalized Gaussian Mixture ---
    data_exp = jnp.expand_dims(data, axis=1) # (N, 1, D_X)
    mu_exp = jnp.expand_dims(mu, axis=0)     # (1, K, D_X)
    sigma_exp = jnp.expand_dims(sigma, axis=0) # (1, K, D_X)

    log_pdf_clusters = jnp.sum(stats.norm.logpdf(data_exp, loc=mu_exp, scale=sigma_exp), axis
    weighted_log_pdf = log_p + log_pdf_clusters # (N, K)

    # LogSumExp merges K clusters back together
    total_log_likelihood = jax.scipy.special.logsumexp(weighted_log_pdf, axis=-1)

    # Target Likelihood
    numpyro.factor('mixture_likelihood', jnp.sum(total_log_likelihood))

# Run MCMC
m.fit(bnn_mixture_model, num_chains=1)
```

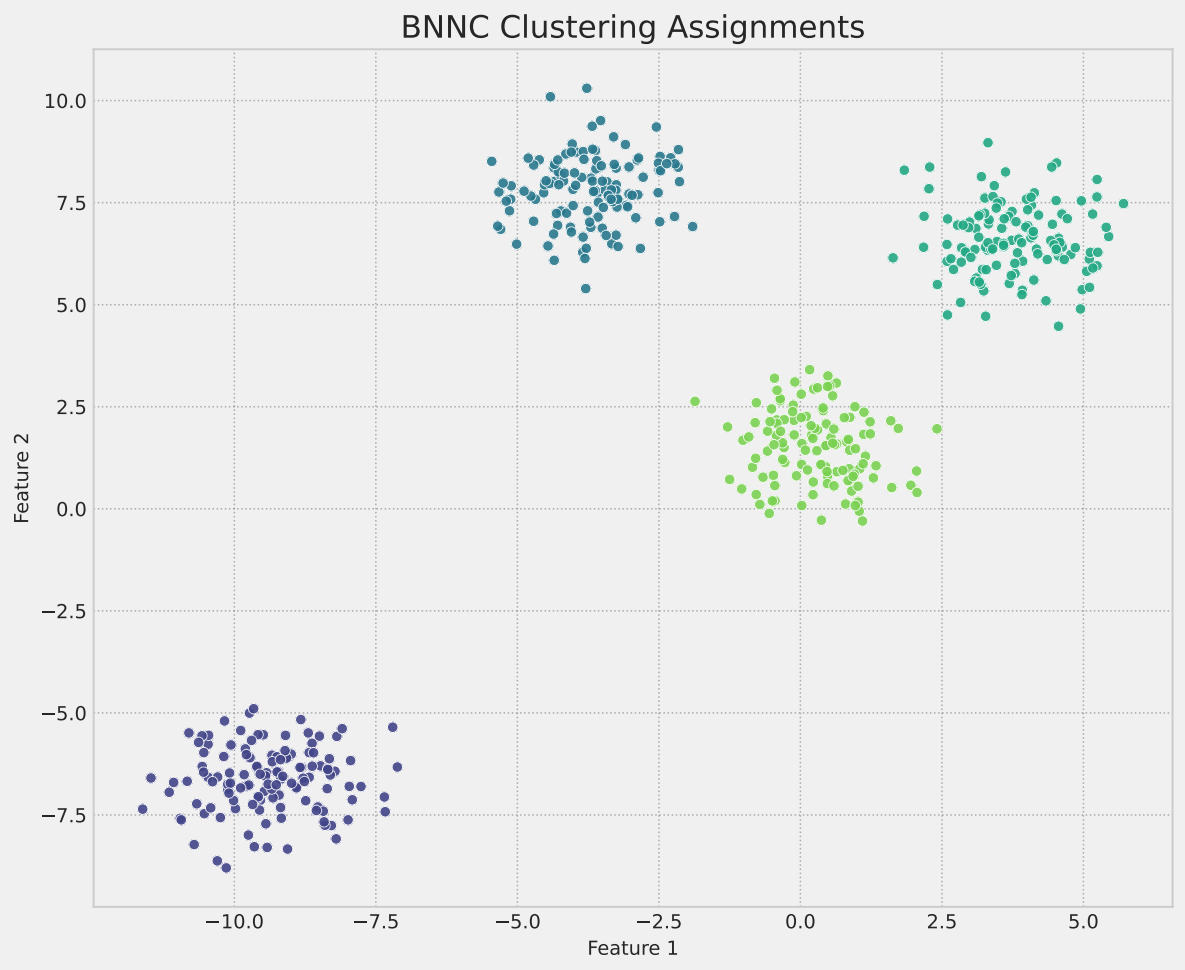
Python (Build in function)

```
from BayesForge import bf
from sklearn.datasets import make_blobs
m = bf()

# Generate Synthetic Data ----------------------------------
data, true_labels = make_blobs(
    n_samples=500, centers=4, cluster_std=0.8,
    center_box=(-10,10), random_state=101
)

# K is the maximum component limit, D_H1 is the hidden width
```

```python
m.data_on_model = dict(data=data, K=11, D_H1=10, empirical_bayes=True) # Enables empirical pr

# Run MCMC
m.fit(m.models.bnnc, num_chains=1)

# Predict cluster targets
theta_samps, mu_samps, sigma_samps, labels = m.models.bnnc.predict(data, m.sampler)

# Plot Results
m.models.bnnc.plot(data, m.sampler)
```

```
/home/sosa/work/3.12venv/lib/python3.10/site-packages/tqdm/auto.py:21: TqdmWarning:

IProgress not found. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs

bf v 0.0.47 package loaded
jax.local_device_count 32
 This function is still in development. Use it with caution.
 This function is still in development. Use it with caution.
 This function is still in development. Use it with caution.
 This function is still in development. Use it with caution.
 This function is still in development. Use it with caution.
 This function is still in development. Use it with caution.
 This function is still in development. Use it with caution.
 This function is still in development. Use it with caution.
 This function is still in development. Use it with caution.

  0%|          | 0/2000 [00:00<?, ?it/s]

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

warmup:   0%|          | 1/2000 [00:00<23:55,  1.39it/s, 1 steps of size 2.34e+00. acc. prob=

 This function is still in development. Use it with caution.
Model found 4 clusters.
```

```
 This function is still in development. Use it with caution.
 This function is still in development. Use it with caution.
Model found 4 clusters.
```



Mathematical Details

In the Bayesian formulation, we place priors on all weights and biases and define a likelihood for the output. For a BNNC model with a K-hidden-layer gating network and a D_X-vector of predictors we can run the model as below. For the code example, we consider a single hidden layer gating network with a hyperbolic tangent (tanh) activation function . Because the input matrix X incorporates the intercept as its first column, the bias term is implicitly included in the layer's weights:

$$
\begin{aligned}
\log p(X_i \mid \mu, \sigma, \theta) &= \text{LogSumExp}\left[\log(\theta_{ik}) + \log \mathcal{N}(X_i \mid \mu_k, \sigma_k)\right]_{k=1}^{K} \\
\theta_i &= \text{Softmax}(\phi_i) \\
\phi_i &= H_i W_2 + \log(\pi) \\
H_i &= \tanh(X_i W_1) \\
\pi &\sim \text{Dirichlet}(\frac{\alpha}{K}, \ldots, \frac{\alpha}{K}) \\
W_1 &\sim \text{Normal}(0, 1) \\
W_2 &\sim \text{Normal}(0, 0.05) \\
\mu_k &\sim \text{Normal}(0, 5) \\
\sigma_k &\sim \text{Exponential}(1)
\end{aligned}
$$

where:

- X_i is the observed data vector for the i-th observation.
- θ_{ik} is the probability of assigning observation i to component k.
- π is the global cluster weight vector, which enforces parsimony via a Dirichlet prior.
- α is the concentration parameter (e.g., 0.05), driving unused clusters towards zero.
- H_i is the hidden layer representation vector for the i-th observation ($D_H = 10$).
- W_1 and W_2 are the weight matrices of the gating network.
- μ_k and σ_k are the mean and standard deviation for the k-th Gaussian mixture component.
- All gating network weights are assigned independent Normal priors.

Reference(s)

1. Murphy, K. P. (2012). *Machine Learning: A Probabilistic Perspective*. MIT Press.