In [47]:

```
import numpy as np
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error,accuracy_score,classification_report
import skfuzzy as fuzz
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
import pandas
```

In [48]:

```
# CHOOSE DATASET

# Regression dataset
#diabetes = datasets.load_diabetes(as_frame=True)

# CLassification dataset
diabetes = datasets.fetch_openml("diabetes",version=1, as_frame=True)

X = diabetes.data.values
y = diabetes.target.values


# Converter labels em binário (0 = negativo, 1 = positivo) (só usado em classification)
y= np.array([1 if val == "tested_positive" else 0 for val in y])

# Converter para tensor PyTorch (coluna) ( só usado em classification)
y = torch.tensor(y, dtype=torch.float32).reshape(-1, 1)


X.shape
```

Out[48]:

```
(768, 8)
```

In [49]:

```
print (y)
```

```
tensor([[1.],
        [0.],
        [1.],
        [0.],
        [1.],
        [0.],
        [1.],
        [0.],
        [1.],
        [1.],
        [0.],
        [1.],
        [0.],
        [1.],
        [1.],
        [1.],
        [1.],
        [1.],
        [0.],
        [1.],
        [0.],
        [0.],
        [1.],
        [1.],
```

```
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
```

```
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
```

```
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
```

```
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
```

```
[1.],
[0.],
[1.],
[0.],
[0.],
[1.],
[0.],
[1.],
[0.],
[1.],
[1.],
[1.],
[0.],
[0.],
[1.],
[0.],
[1.],
[0.],
[0.],
[0.],
[1.],
[0.],
[0.],
[0.],
[0.],
[1.],
[1.],
[1.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[1.],
[0.],
[0.],
[0.],
[0.],
[0.],
[1.],
[1.],
[1.],
[0.],
[1.],
[1.],
[0.],
[0.],
[1.],
[0.],
[0.],
[1.],
[0.],
[0.],
[1.],
[1.],
[0.],
[0.],
[0.],
[0.],
[1.],
[0.],
[0.],
[1.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
```

```
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
```

```
             [0.],
             [0.],
             [1.],
             [0.],
             [0.],
             [0.],
             [0.],
             [0.],
             [0.],
             [0.],
             [0.],
             [0.],
             [1.],
             [0.],
             [0.],
             [0.],
             [0.],
             [0.],
             [0.],
             [0.],
             [1.],
             [0.],
             [0.],
             [0.],
             [1.],
             [0.],
             [0.],
             [0.],
             [1.],
             [1.],
             [0.],
             [0.],
             [0.],
             [0.],
             [0.],
             [0.],
             [0.],
             [1.],
             [0.],
             [0.],
             [0.],
             [0.],
             [1.],
             [0.],
             [0.],
             [0.],
             [1.],
             [0.],
             [0.],
             [0.],
             [1.],
             [0.],
             [0.],
             [0.],
             [1.],
             [0.],
             [0.],
             [0.],
             [0.],
             [1.],
             [1.],
             [0.],
             [0.],
             [0.],
             [0.],
             [0.],
             [0.],
             [1.],
             [0.],
             [0.],
             [0.],
             [0.],
             [0.
```

```
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[1.],
[0.],
[0.],
[0.],
[1.],
[1.],
[1.],
[1.],
[0.],
[0.],
[1.],
[1.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[1.],
[1.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[1.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[1.],
[0.],
[1.],
[1.],
[0.],
[0.],
[0.],
[1.],
[0.],
[1.],
[0.],
[1.],
[0.],
[1.],
[0.],
[1.],
[0.],
[0.],
[1.],
[0.],
[0.],
[1.],
[0.],
```

```
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
```

```
[0.],
[0.],
[0.],
[1.],
[1.],
[0.],
[1.],
[0.],
[0.],
[1.],
[0.],
[1.],
[0.],
[0.],
[0.],
[0.],
[0.],
[1.],
[0.],
[1.],
[0.],
[1.],
[0.],
[1.],
[1.],
[0.],
[0.],
[0.],
[0.],
[1.],
[1.],
[0.],
[0.],
[0.],
[1.],
[0.],
[1.],
[1.],
[0.],
[0.],
[1.],
[0.],
[0.],
[1.],
[1.],
[0.],
[0.],
[1.],
[0.],
[0.],
[1.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[1.],
[1.],
[1.],
[0.],
[0.],
[0.],
[0.],
[0.],
[0.],
[1.],
[1.],
[0.],
[0.],
[1.],
[0.],
```

```
       [0.],
       [0.],
       [1.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [0.],
       [1.],
       [1.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [1.],
       [0.],
       [0.],
       [0.],
       [0.],
       [1.],
       [0.]])
```

In [50]:

```
#train test spliting
test_size=0.2
Xtr, Xte, ytr, yte = train_test_split(X, y, test_size=test_size, random_state=42)
```

In [51]:

```
# Standardize features
scaler=StandardScaler()
Xtr= scaler.fit_transform(Xtr)
Xte= scaler.transform(Xte)
```

In [52]:

```
# Number of clusters
n_clusters = 2
m=6.75

# Concatenate target for clustering
Xexp=np.concatenate([Xtr, ytr.reshape(-1, 1)], axis=1)
#Xexp=Xtr

# Transpose data for skfuzzy (expects features x samples)
Xexp_T = Xexp.T

# Fuzzy C-means clustering
centers, u, u0, d, jm, p, fpc = fuzz.cluster.cmeans(
    Xexp_T, n_clusters, m=m, error=0.005, maxiter=1000, init=None,
)
```

In [53]:

```
centers.shape
```

Out[53]:

```
(2, 9)
```

In [54]:

```
# Compute sigma (spread) for each cluster
sigmas = []
for j in range(n_clusters):
    # membership weights for cluster j, raised to m
    u_j = u[j, :] ** m
```

```
    # weighted variance for each feature
    var_j = np.average((Xexp - centers[j])**2, axis=0, weights=u_j)
    sigma_j = np.sqrt(var_j)
    sigmas.append(sigma_j)
sigmas=np.array(sigmas)
```

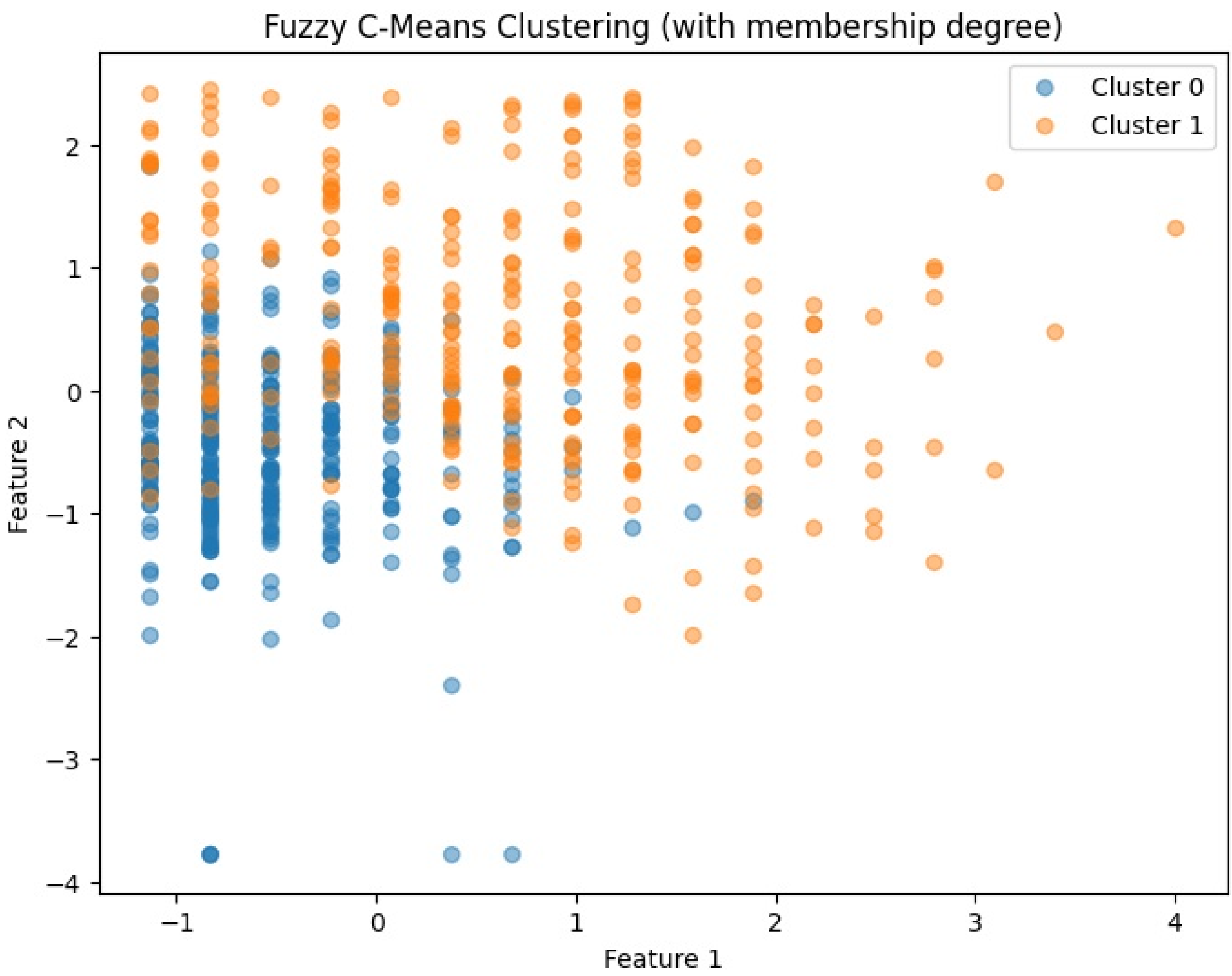
In [55]:

```
# Hard clustering from fuzzy membership
cluster_labels = np.argmax(u, axis=0)
print("Fuzzy partition coefficient (FPC):", fpc)

# Plot first two features with fuzzy membership
plt.figure(figsize=(8,6))
for j in range(n_clusters):
    plt.scatter(
        Xexp[cluster_labels == j, 0],              # Feature 1
        Xexp[cluster_labels == j, 1],              # Feature 2
        alpha=u[j, :],           # transparency ~ membership
        label=f'Cluster {j}'
    )

plt.title("Fuzzy C-Means Clustering (with membership degree)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```

```
Fuzzy partition coefficient (FPC): 0.5000000339104199
```

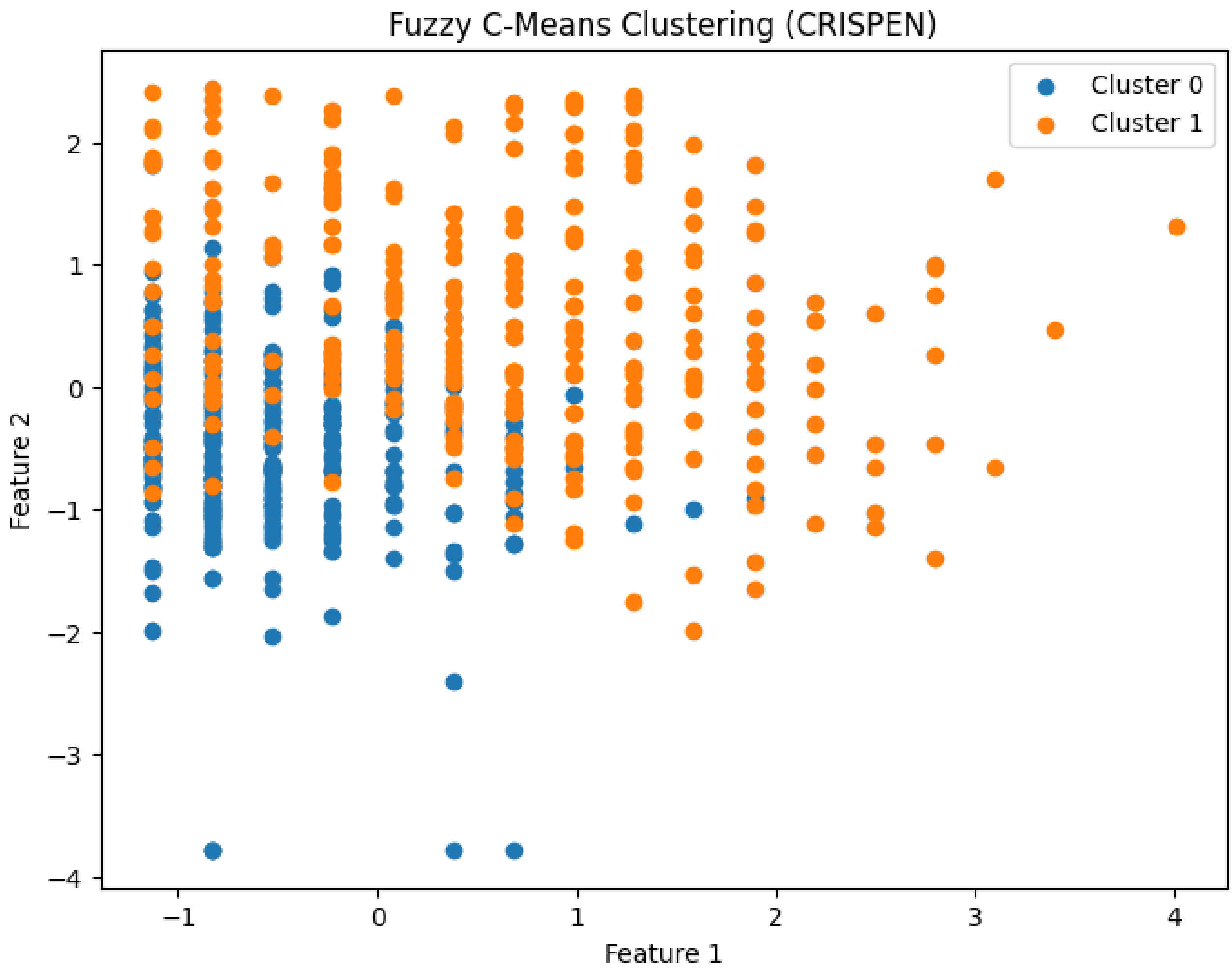


In [56]:

```
# Plot first two features with cluster assignments
plt.figure(figsize=(8,6))
for j in range(n_clusters):
    plt.scatter(
        Xexp[cluster_labels == j, 0],
        Xexp[cluster_labels == j, 1],
```

```
        label=f'Cluster {j}'
    )

plt.title("Fuzzy C-Means Clustering (CRISPEN)")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.legend()
plt.show()
```



In [57]:

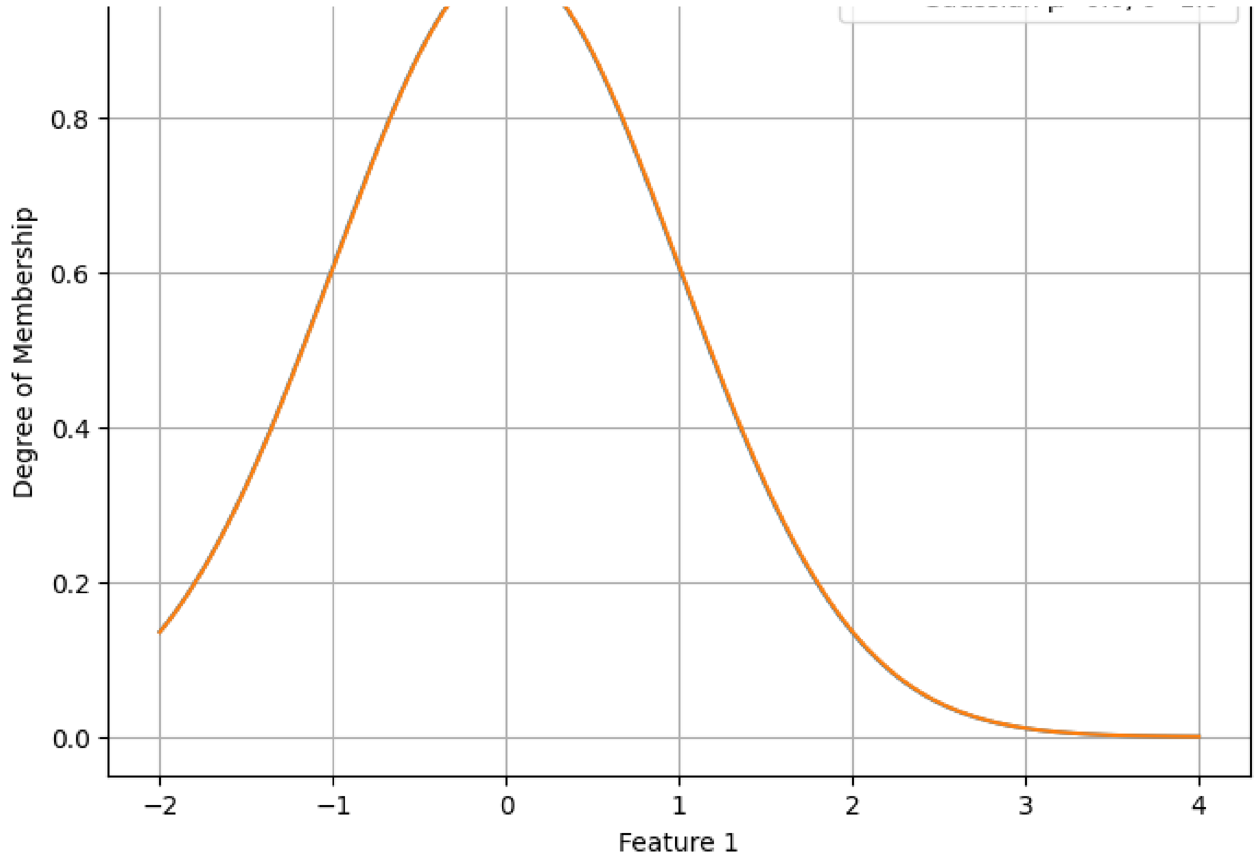
```
# Gaussian formula
def gaussian(x, mu, sigma):
    return np.exp(-0.5 * ((x - mu)/sigma)**2)

lin=np.linspace(-2, 4, 500)
plt.figure(figsize=(8,6))

y_aux=[]
for j in range(n_clusters):
# Compute curves
    y_aux.append(gaussian(lin, centers[j,0], sigmas[j,0]))

# Plot
    plt.plot(lin, y_aux[j], label=f"Gaussian μ={np.round(centers[j,0],2)}, σ={np.round(si
gmas[j,0],2)}")

plt.title("Projection of the membership functions on Feature 2")
plt.xlabel("Feature 1")
plt.ylabel("Degree of Membership")
plt.legend()
plt.grid(True)
plt.show()
```



In [58]:

```
# ---------------------------
# Gaussian Membership Function
# ---------------------------
class GaussianMF(nn.Module):
    def __init__(self, centers, sigmas, agg_prob):
        super().__init__()
        self.centers = nn.Parameter(torch.tensor(centers, dtype=torch.float32))
        self.sigmas = nn.Parameter(torch.tensor(sigmas, dtype=torch.float32))
        self.agg_prob=agg_prob

    def forward(self, x):
        # Expand for broadcasting
        # x: (batch, 1, n_dims), centers: (1, n_rules, n_dims), sigmas: (1, n_rules, n_di
ms)
        diff = abs((x.unsqueeze(1) - self.centers.unsqueeze(0))/self.sigmas.unsqueeze(0)
) #(batch, n_rules, n_dims)

        # Aggregation
        if self.agg_prob:
            dist = torch.norm(diff, dim=-1)  # (batch, n_rules) # probablistic intersect
ion
        else:
            dist = torch.max(diff, dim=-1).values  # (batch, n_rules) # min intersection
(min instersection of normal funtion is the same as the max on dist)

        return torch.exp(-0.5 * dist ** 2)


# ---------------------------
# TSK Model
# ---------------------------
class TSK(nn.Module):
    def __init__(self, n_inputs, n_rules, centers, sigmas,agg_prob=False):
        super().__init__()
        self.n_inputs = n_inputs
        self.n_rules = n_rules

        # Antecedents (Gaussian MFs)

        self.mfs=GaussianMF(centers, sigmas,agg_prob)
```

```
        # Consequents (linear functions of inputs)
        # Each rule has coeffs for each input + bias
        self.consequents = nn.Parameter(
            torch.randn(n_inputs + 1,n_rules)
        )

    def forward(self, x):
        # x: (batch, n_inputs)
        batch_size = x.shape[0]

        # Compute membership values for each input feature
        # firing_strengths: (batch, n_rules)
        firing_strengths = self.mfs(x)

        # Normalize memberships
        # norm_fs: (batch, n_rules)
        norm_fs = firing_strengths / (firing_strengths.sum(dim=1, keepdim=True) + 1e-9)

        # Consequent output (linear model per rule)
        x_aug = torch.cat([x, torch.ones(batch_size, 1)], dim=1)   # add bias

        rule_outputs = torch.einsum("br,rk->bk", x_aug, self.consequents)   # (batch, rul
es)
        # Weighted sum
        output = torch.sum(norm_fs * rule_outputs, dim=1, keepdim=True)

        return output, norm_fs, rule_outputs
```

In [59]:

```
# ---------------------------
# Least Squares Solver for Consequents (TSK)
# ---------------------------
def train_ls(model, X, y):
    with torch.no_grad():
        _, norm_fs, _ = model(X)

        # Design matrix for LS: combine normalized firing strengths with input
        X_aug = torch.cat([X, torch.ones(X.shape[0], 1)], dim=1)

        Phi = torch.einsum("br,bi->bri", X_aug, norm_fs).reshape(X.shape[0], -1)

        # Solve LS: consequents = (Phi^T Phi)^-1 Phi^T y

        theta= torch.linalg.lstsq(Phi, y).solution


        model.consequents.data = theta.reshape(model.consequents.shape)
```

In [60]:

```
# ---------------------------
# Gradient Descent Training
# ---------------------------
def train_gd(model, X, y, epochs=100, lr=1e-3):
    optimizer = optim.Adam(model.parameters(), lr=lr)
    criterion = nn.MSELoss()
    for _ in range(epochs):
        optimizer.zero_grad()
        y_pred, _, _ = model(X)
        loss = criterion(y_pred, y)
        print(loss)
        loss.backward()
        optimizer.step()
```

In [61]:

```
# ---------------------------
# Hybrid Training (Classic ANFIS)
# ---------------------------
```

```python
def train_hybrid_alternating(model, X, y, max_iters=10, gd_epochs=20, lr=1e-3):
    train_ls(model, X, y)
    for _ in range(max_iters):
        # Step A: GD on antecedents (freeze consequents)
        for p in model.consequents.parameters():
            p.requires_grad = False
        train_gd(model, X, y, epochs=gd_epochs, lr=lr)

        # Step B: LS on consequents (freeze antecedents)
        for p in model.consequents.parameters():
            p.requires_grad = True
        for p in model.mfs.parameters():
            p.requires_grad = False
        train_ls(model, X, y)

        # Re-enable antecedents
        for p in model.mfs.parameters():
            p.requires_grad = True
```

In [62]:

```python
# ---------------------------
# Alternative Hybrid Training (LS+ gradient descent on all)
# ---------------------------
def train_hybrid_classic(model, X, y, epochs=100, lr=1e-4):
    # Step 1: LS for consequents
    train_ls(model, X, y)
    # Step 2: GD fine-tuning
    train_gd(model, X, y, epochs=epochs, lr=lr)
```

In [63]:

```python
# Build model
model = TSK(n_inputs=Xtr.shape[1], n_rules=n_clusters, centers=centers[:,:-1], sigmas=si
gmas[:,:-1])

Xtr = torch.tensor(Xtr, dtype=torch.float32)
ytr = torch.tensor(ytr, dtype=torch.float32)
Xte = torch.tensor(Xte, dtype=torch.float32)
yte = torch.tensor(yte, dtype=torch.float32)
```

```
C:\Users\banan\AppData\Local\Temp\ipykernel_39380\2256170614.py:5: UserWarning: To copy c
onstruct from a tensor, it is recommended to use sourceTensor.detach().clone() or sourceT
ensor.detach().clone().requires_grad_(True), rather than torch.tensor(sourceTensor).
  ytr = torch.tensor(ytr, dtype=torch.float32)
C:\Users\banan\AppData\Local\Temp\ipykernel_39380\2256170614.py:7: UserWarning: To copy c
onstruct from a tensor, it is recommended to use sourceTensor.detach().clone() or sourceT
ensor.detach().clone().requires_grad_(True), rather than torch.tensor(sourceTensor).
  yte = torch.tensor(yte, dtype=torch.float32)
```

In [64]:

```python
# Training with LS:
train_ls(model, Xtr, ytr.reshape(-1,1))
```

In [65]:

```python
y_pred, _, _=model(Xte)
#performance metric for classification
print(f'ACC:{accuracy_score(yte.detach().numpy(),y_pred.detach().numpy()>0.5)}') #classif
ication
#performance metric for regression
#print(f'MSE:{mean_squared_error(yte.detach().numpy(),y_pred.detach().numpy())}') #regres
sion
```

```
ACC:0.7597402597402597
```