In [1]:
```python
import torch
from kan import KAN

import numpy as np
import sympy as sp
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
```

通过利用模拟数据对KAN网络进行了探索和实验，以验证其在分类任务中的表现。KAN网络作为一种结合了知识图谱和深度学习技术的模型，在处理复杂关系和多模态数据方面展现出巨大的潜力。

模拟数据的优势在于其可控性和灵活性。通过设计特定的规则和参数，我们可以生成具有明确特征和标签的数据集，从而更好地理解模型在不同条件下的行为和性能。此外，模拟数据还可以帮助我们避免实际数据可能带来的隐私和安全问题，使得实验过程更加便捷和高效。

生成模拟数据

In [2]:
```python
from sklearn.datasets import make_classification

n_samples = 1000
n_features = 20
n_classes = 2
random_state = 42

# 生成数据
X, y = make_classification(n_samples=n_samples,
                           n_features=n_features,
                           n_classes=n_classes,
                           random_state=random_state)

# 将数据转换为DataFrame查看
df = pd.DataFrame(X, columns=[f'feature_{i}' for i in range(n_features)])
df['target'] = y

df.head()
```

Out[2]:

	feature_0	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feat
0	-0.669356	-1.495778	-0.870766	1.141831	0.021606	1.730630	-1.251698	0.2
1	0.093372	0.785848	0.105754	1.272354	-0.846316	-0.979093	1.263707	0.2
2	-0.905797	-0.608341	0.295141	0.943716	0.092936	1.370397	-0.064772	0.2
3	-0.585793	0.389279	0.698816	0.436236	-0.315082	0.459505	1.448820	0.5
4	1.146441	0.515579	-1.222895	-0.396230	-1.293508	-0.352428	0.071254	1.2

5 rows × 21 columns

定义设备

In [3]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda
```

划分数据集

In [4]:
```
X_train, X_test, y_train, y_test = train_test_split(df.drop(["target"],axis=1),
                                                    df["target"],
                                                    test_size=0.30,
                                                    random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_test,
                                                y_test,
                                                test_size=0.6,
                                                random_state=42)

# Covert data to torch tensor
train_input = torch.tensor(X_train.to_numpy(), dtype=torch.float32).to(device)
train_label = torch.tensor(y_train.to_numpy(), dtype=torch.long).to(device)
val_input = torch.tensor(X_val.to_numpy(), dtype=torch.float32).to(device)
val_label = torch.tensor(y_val.to_numpy(), dtype=torch.long).to(device)
test_input = torch.tensor(X_test.to_numpy(), dtype=torch.float32).to(device)
test_label = torch.tensor(y_test.to_numpy(), dtype=torch.long).to(device)

dataset = {
    'train_input': train_input,
    'train_label': train_label,
    'val_input': val_input,
    'val_label': val_label,
    'test_input': test_input,
    'test_label': test_label
}
```

构建分类模型

In [5]:
```
model = KAN(width=[X_train.shape[1], 7, 2], grid=5, k=11, seed=42, device=device
```

```python
dtype=torch.float32
# 准确率
def train_acc():
    return torch.mean((torch.argmax(model(dataset['train_input']), dim=1) == dat

def test_acc():
    return torch.mean((torch.argmax(model(dataset['test_input']), dim=1) == data

results = model.fit(dataset, opt="LBFGS", steps=20, metrics=(train_acc, test_acc
results['train_acc'][-1], results['test_acc'][-1]
```

```
checkpoint directory created: ./model
saving model version 0.0
| train_loss: 2.43e-02 | test_loss: 2.15e+00 | reg: 1.43e+02 | : 100%|█| 20/20 [0
0:12<00:00,  1.65it
saving model version 0.1
```

Out[5]:

```
(1.0, 0.7944444417953491)
```

评估

f1

In [6]:

```python
from sklearn.metrics import f1_score

model.eval()

with torch.no_grad():  # 确保不会追踪梯度，节省内存和计算资源
    # Test dataset evaluation
    test_preds = model(test_input)
    test_plabels = torch.argmax(test_preds, dim=1).cpu().numpy()  # 转换为类别标
    test_labels = test_label.cpu().numpy()

    # Train dataset evaluation
    train_preds = model(train_input)
    train_plabels = torch.argmax(train_preds, dim=1).cpu().numpy()
    train_labels = train_label.cpu().numpy()

    # Validation dataset evaluation
    val_preds = model(val_input)
    val_plabels = torch.argmax(val_preds, dim=1).cpu().numpy()
    val_labels = val_label.cpu().numpy()

# Evaluate metrics
print("Train f1_score:", f1_score(train_labels, train_plabels, average='weighted

print("Val f1_score:", f1_score(val_labels, val_plabels, average='weighted'))

print("Test f1_score:", f1_score(test_labels, test_plabels, average='weighted'))
```

```
Train f1_score: 1.0
Val f1_score: 0.8164620535714285
Test f1_score: 0.7946931697916774
```

评估函数

In [7]:

```python
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score
)

def scoring_clf(y_true, y_pred):
    print("-" * 10, "\n")
    print("accuracy_score:", accuracy_score(y_true, y_pred))
    print("precision_score:", precision_score(y_true, y_pred))
    print("recall_score:", recall_score(y_true, y_pred))
    print("f1_score:", f1_score(y_true, y_pred))

model.eval()

with torch.no_grad():  # 确保不会追踪梯度，节省内存和计算资源
    # Test dataset evaluation
    test_preds = model(test_input)
    test_plabels = torch.argmax(test_preds, dim=1).cpu().numpy()  # 转换为类别标
    test_labels = test_label.cpu().numpy()

    # Train dataset evaluation
    train_preds = model(train_input)
    train_plabels = torch.argmax(train_preds, dim=1).cpu().numpy()
    train_labels = train_label.cpu().numpy()

    # Validation dataset evaluation
    val_preds = model(val_input)
    val_plabels = torch.argmax(val_preds, dim=1).cpu().numpy()
    val_labels = val_label.cpu().numpy()


scoring_clf(train_labels, train_plabels)
scoring_clf(val_labels, val_plabels)
scoring_clf(test_labels, test_plabels)
```

```
----------

accuracy_score: 1.0
precision_score: 1.0
recall_score: 1.0
f1_score: 1.0
----------

accuracy_score: 0.8166666666666667
precision_score: 0.8181818181818182
recall_score: 0.7894736842105263
f1_score: 0.8035714285714286
----------

accuracy_score: 0.7944444444444444
precision_score: 0.8210526315789474
recall_score: 0.7959183673469388
f1_score: 0.8082901554404145
```