1. 非线性拟合实验报告

Author : Xinyu Zhao,Beihang University
Email:zxy2021@buaa.edu.cn

摘要

本报告详细探讨了使用多项式回归、神经网络等多种方法进行非线性拟合的过程。我们的目标是找到一个能够最佳拟合给定训练集的模型，同时评估其在独立测试集上的性能。通过对不同非线性模型的比较、模型训练过程的优化、以及深入的结构分析，我们旨在提高模型的预测准确性和泛化能力。

1.1 问题描述

给定范围为0~10、均匀分布的100组二维数据点组成的训练集和测试集，我们采用**均方误差(MAE)**作为评价指标，旨在通过非线性拟合方法捕捉数据中的复杂关系。本实验首先尝试了多项式回归，并随后探索了使用神经网络进行拟合。

均方误差(MAE)是机器学习中较为常见的一种误差指标，对应于二次损失函数，使得计算和数学推导变得更加简单。在优化问题中，使用均方误差可以导出解析解，这是因为对于线性模型，二次损失函数的导数是线性的，这使得找到最小值变得相对容易。因此在本次实验中定义的评价指标为**均方误差**。

1.2 数据描述

数据集和测试集中分别包含范围为0~10且均匀分布的100组数据，作图如下，可以看出数据均为复杂的非线性分布。

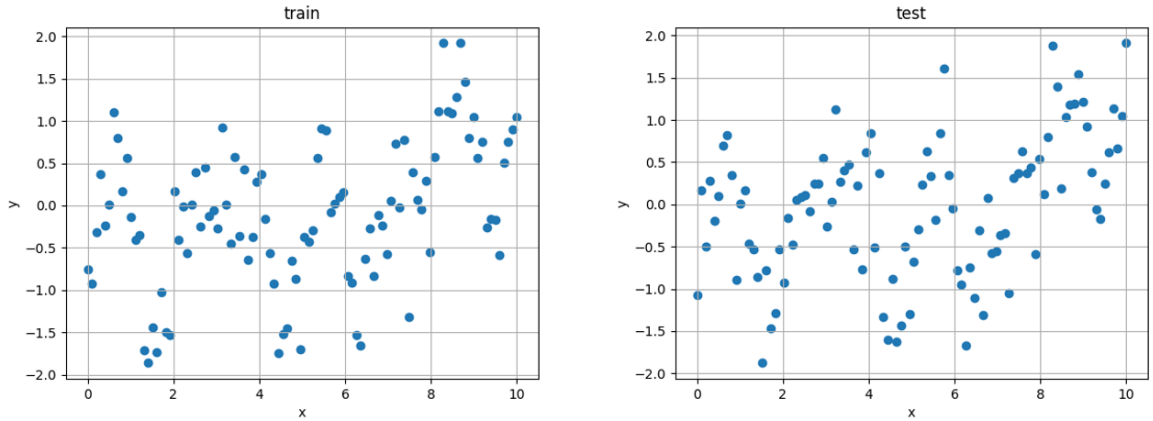

图1 原始训练/测试集的散点图

```python
#绘图代码如下：
import pandas as pd
import matplotlib.pyplot as plt

# 读取数据
df = pd.read_excel("train.xlsx", usecols=[0, 1])

# 提取x和y
x = df.iloc[:, 0]
y = df.iloc[:, 1]

# 绘制散点图
plt.scatter(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.title('train')
plt.grid(True)
plt.show()
```

1.3 模型建立与拟合

1.3.1 多项式回归

我们从简单的多项式回归开始，逐渐增加多项式的阶数以提高拟合度。通过比较不同阶数下的MAE，我们发现15阶多项式提供了最佳的拟合效果，最小误差值为0.44841866097882854。在1~15阶的过程中，随着次数增加，可拟合度增加，提高了拟合效果，而当阶数大于15阶之后，拟合函数在训练集上逐渐表现为过拟合，导致在测试数据集上面表现不理想，因而损失值上升。下面展示部分多项式阶数（5,8,10,15,20,35,50,65）的拟合效果（其中蓝点为训练集、红点为测试集）：

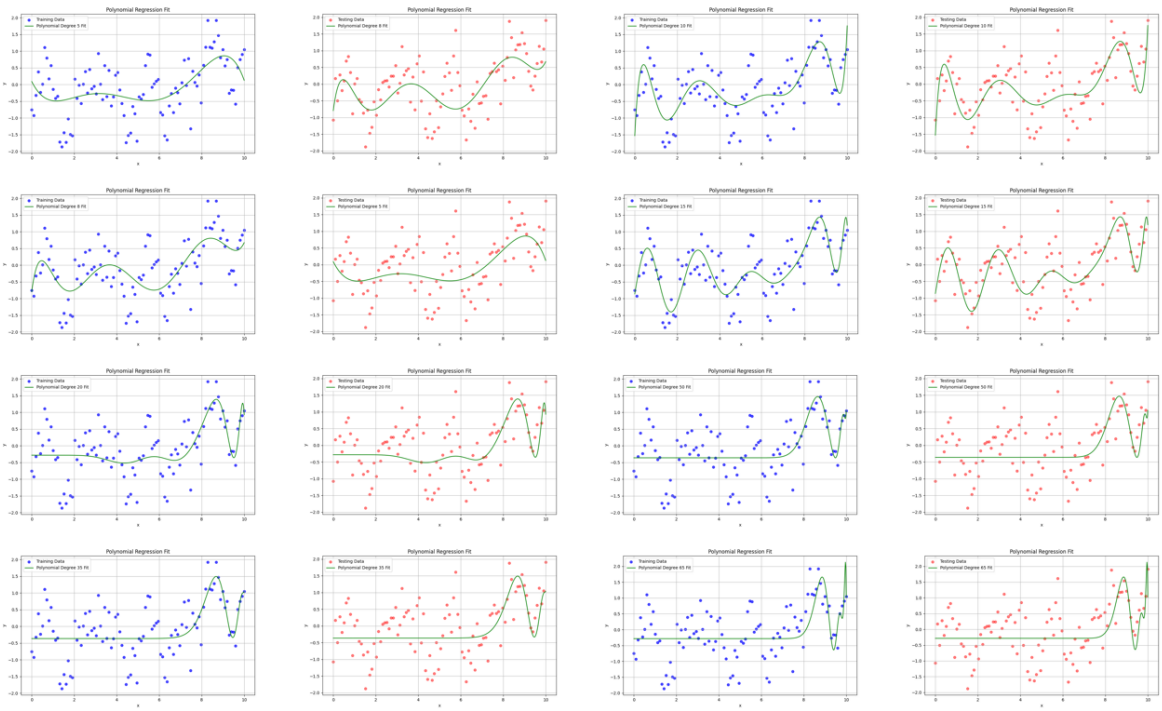

图3 部分阶数拟合效果图

```python
# 关键代码如下
for degree_best in range(1,101):
    poly_model_best = make_pipeline(PolynomialFeatures(degree_best),
LinearRegression())
```

```
    poly_model_best.fit(X_train, y_train)
    y_pred_best = poly_model_best.predict(X_test)

    # 计算 MAE
    loss_best = mean_absolute_error(y_test, y_pred_best)
    print(f'Best Model (Degree {degree_best}) MAE:', loss_best)
    loss_list.append(loss_best)
    # 创建用于绘图的预测范围
    x_range = np.linspace(X_train['x'].min(), X_train['x'].max(), 500)
    y_range_pred = poly_model_best.predict(x_range.reshape(-1, 1))

    if degree_best in [5,8,10,15,20,35,50,65]:
        # 训练集绘图
        plt.figure(figsize=[10, 6])
        plt.scatter(X_train['x'], y_train, color='blue', alpha=0.7,
label='Training Data')
        plt.plot(x_range, y_range_pred, color='green', label=f'Polynomial
Degree {degree_best} Fit')
        plt.title('Polynomial Regression Fit')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.legend()
        plt.grid(True)
        plt.show()

        # 测试集绘图
        plt.figure(figsize=[10, 6])
        plt.scatter(X_test['x'], y_test, color='red', alpha=0.5,
label='Testing Data')
        plt.plot(x_range, y_range_pred, color='green', label=f'Polynomial
Degree {degree_best} Fit')
        plt.title('Polynomial Regression Fit')
        plt.xlabel('x')
        plt.ylabel('y')
        plt.legend()
        plt.grid(True)
        plt.show()
```

将每次的损失值存入列表loss_list中，绘制趋势图如下：

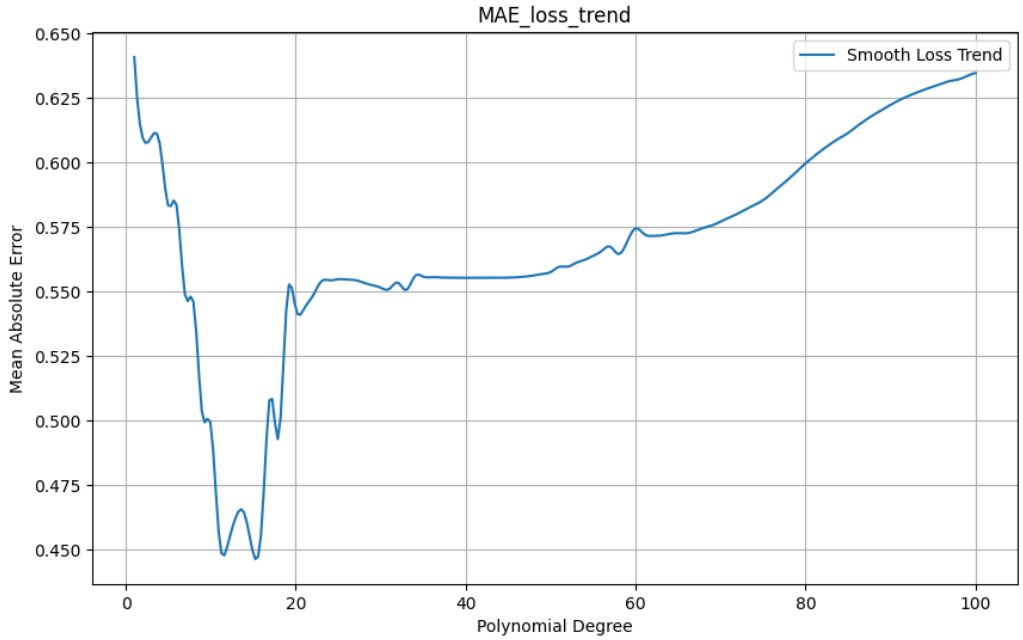

图4 损失值与多项式阶数之间的关系

代码如下：

```
# 创建一个索引列表，用作X轴数据
x_values = np.array(range(1, len(loss_list) + 1))
y_values = np.array(loss_list)

# 创建一个平滑的曲线
x_smooth = np.linspace(x_values.min(), x_values.max(), 300)  # 生成足够多的x值以使曲线平滑
spl = make_interp_spline(x_values, y_values, k=3)  # k是样条曲线的阶数，这里使用三次样条
y_smooth = spl(x_smooth)

# 使用matplotlib绘制平滑的曲线
plt.figure(figsize=[10, 6])
plt.plot(x_smooth, y_smooth, label='Smooth Loss Trend')  # 平滑曲线
plt.title('MAE_loss_trend')
plt.xlabel('Polynomial Degree')
plt.ylabel('Mean Absolute Error')
plt.legend()
plt.grid(True)
plt.show()
```

1.3.2 神经网络拟合

为了进行拟合模型之间的性能比较，本文提出利用神经网络进行拟合，以提高模型的灵活性和准确性，关键代码如下：

```
# 数据读取
train_data = pd.read_excel('train.xlsx')
test_data = pd.read_excel('test.xlsx')
X_train = train_data['x'].values.reshape(-1, 1)
Y_train = train_data['y'].values.reshape(-1, 1)
```

```python
X_test = test_data['x'].values.reshape(-1, 1)
Y_test = test_data['y'].values.reshape(-1, 1)
# 定义神经网络结构
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()
        self.fc1 = nn.Linear(1, 64)  # 输入层到第一个隐藏层
        self.fc2 = nn.Linear(64, 64) # 第二个隐藏层
        self.fc3 = nn.Linear(64, 1)  # 输出层

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = torch.relu(self.fc2(x))
        return self.fc3(x)

# 初始化模型、损失函数和优化器
model = SimpleNN()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# 转换数据为torch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
Y_train_tensor = torch.tensor(Y_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)

# 训练模型
epochs = 1000
for epoch in range(epochs):
    optimizer.zero_grad()   # 清除之前的梯度
    output = model(X_train_tensor)  # 前向传播
    loss = criterion(output, Y_train_tensor)  # 计算损失
    loss.backward()         # 反向传播，计算梯度
    optimizer.step()        # 更新权重

    if epoch % 100 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}')

# 使用训练好的模型进行预测
model.eval()  # 设置模型为评估模式
with torch.no_grad():
    Y_pred_nn = model(X_test_tensor).view(-1).numpy()

# 计算平均绝对误差
mae_nn = mean_absolute_error(Y_test, Y_pred_nn)
print(f'Test MAE of the neural network: {mae_nn}')

'''
Epoch [1/1000], Loss: 0.7944
Epoch [101/1000], Loss: 0.5523
Epoch [201/1000], Loss: 0.5039
Epoch [301/1000], Loss: 0.5000
Epoch [401/1000], Loss: 0.4995
Epoch [501/1000], Loss: 0.4992
Epoch [601/1000], Loss: 0.4988
Epoch [701/1000], Loss: 0.4983
Epoch [801/1000], Loss: 0.4978
```

```
Epoch [901/1000], Loss: 0.4973
Test MAE of the neural network: 0.5818524205386759
'''
```

发现运行结果得到的误差为0.5818524205386759，不如15阶多项式模型的拟合结果，推测原因如下：

- **神经网络结构不适当**：如果神经网络的规模（层数或每层的神经元数量）太小，它可能没有足够的能力（表达能力）来捕捉数据中的复杂关系；15阶多项式由于其高度的非线性和复杂性，能够很好地拟合或过拟合某些类型的数据。
- **迭代次数不足**：如果训练神经网络的迭代次数（epoch数量）不够，网络可能还没有足够的时间来学习数据中的模式。
- **数据量**：相比于多项式拟合，神经网络通常需要更多的数据来有效学习。
- **优化器和学习率**：选择不恰当的优化算法或设置不合适的学习率导致神经网络训练不充分。
- **过拟合**：神经网络可能过度学习训练数据中的噪声，而不是数据的真实模式，导致在测试数据上的表现不佳。
- **欠拟合**：相反，如果神经网络太简单，不能捕捉数据中的所有相关模式，也会导致拟合效果不佳。

为此，接下来将代码进行了改进，优化神经网络结构，调整训练次数和学习率。

```python
# 定义改进的神经网络结构
class ImprovedNN(nn.Module):
    def __init__(self):
        super(ImprovedNN, self).__init__()
        self.fc1 = nn.Linear(1, 128)  # 输入层到第一个隐藏层，增加神经元数量
        self.dropout1 = nn.Dropout(0.25)  # 添加Dropout层以减少过拟合
        self.fc2 = nn.Linear(128, 128)  # 第二个隐藏层
        self.dropout2 = nn.Dropout(0.25)
        self.fc3 = nn.Linear(128, 1)  # 输出层

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.dropout1(x)
        x = torch.relu(self.fc2(x))
        x = self.dropout2(x)
        return self.fc3(x)

# 初始化模型、损失函数和优化器
model = ImprovedNN()
criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.01)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1000, gamma=0.9)  # 学习率调度器

# 转换数据为torch tensors
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
Y_train_tensor = torch.tensor(Y_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)

# 训练模型
epochs = 10000
```

```python
for epoch in range(epochs):
    optimizer.zero_grad()  # 清除之前的梯度
    output = model(X_train_tensor)  # 前向传播
    loss = criterion(output, Y_train_tensor)  # 计算损失
    loss.backward()  # 反向传播，计算梯度
    optimizer.step()  # 更新权重
    scheduler.step()  # 更新学习率

    if epoch % 1000 == 0:
        print(f'Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}, LR: {scheduler.get_last_lr()[0]}')

# 使用训练好的模型进行预测
model.eval()  # 设置模型为评估模式
with torch.no_grad():
    Y_pred_nn = model(X_test_tensor).view(-1).numpy()

# 计算平均绝对误差
mae_nn = mean_absolute_error(Y_test, Y_pred_nn)
print(f'Test MAE of the neural network: {mae_nn}')
'''
Epoch [1/10000], Loss: 2.4662, LR: 0.01
Epoch [1001/10000], Loss: 0.3971, LR: 0.009000000000000001
Epoch [2001/10000], Loss: 0.4269, LR: 0.008100000000000001
Epoch [3001/10000], Loss: 0.3931, LR: 0.007290000000000001
Epoch [4001/10000], Loss: 0.4117, LR: 0.006561000000000002
Epoch [5001/10000], Loss: 0.3081, LR: 0.005904900000000002
Epoch [6001/10000], Loss: 0.2981, LR: 0.005314410000000002
Epoch [7001/10000], Loss: 0.3212, LR: 0.004782969000000002
Epoch [8001/10000], Loss: 0.3076, LR: 0.004304672100000002
Epoch [9001/10000], Loss: 0.3468, LR: 0.003874204890000002
Test MAE of the neural network: 0.4521120688577277
'''
```

我们可以明显地发现，拟合效果要好于优化前的神经网络，误差值仅为0.45。

通过深入的模型比较和结构分析，我们得到了关于非线性拟合最有效方法的见解。我们发现，尽管多项式回归简单且易于实现，但在处理复杂数据时，神经网络提供了更高的灵活性和准确性。此外，模型优化和正则化技术的应用对于防止过拟合至关重要。

1.4 结论

本报告展示了非线性拟合在复杂数据集上的应用，并比较了多种方法的性能，结果强调了选择合适模型、优化训练过程和深入理解模型结构的重要性。以后将探索更多的非线性拟合技术，并在更广泛的数据集上验证这些方法的有效性。

1.5 附录

实验所用代码、图片以及数据集后续会上传到github上。