2025 中国研究生“华为杯”数学建模竞赛 E 题——GPT 思路

E 题：高速列车轴承智能故障诊断问题

随着高速列车的广泛应用，轴承故障成为影响列车安全运营的重要因素。由于轴承在高速列车中的关键作用，一旦发生故障，轻则导致列车晚点，重则可能引发脱轨等重大事故，严重危及乘客安全。然而，传统的轴承故障诊断方法依赖于专家经验和基于规则的信号处理技术，难以应对复杂的运行环境，尤其是在数据稀缺或样本不平衡的情况下，故障诊断的准确性和实时性受到较大限制。因此，基于大数据和人工智能的智能故障诊断方法在轨道交通领域逐渐获得关注。本研究致力于通过迁移学习技术，提高高速列车轴承故障的诊断准确性和实时性。

问题 1 数据分析与故障特征提取。针对源域实验数据，本文首先**分析了轴承故障的特征，并利用时域、频域及时频域的方法对数据进行特征提取。通过建立轴承故障与各类工作状态（如内圈故障、外圈故障等）之间的关联模型，为后续故障诊断任务提供了基础数据支持。**通过特征分析，明确了轴承故障的关键特征，从而为后续的模型训练和迁移学习提供了理论依据和数据基础。

问题 2 源域故障诊断。在源域数据上，本文**设计并训练了基于深度学习的故障诊断模型。通过划分源域数据的训练集与测试集，建立了轴承故障的分类模型，并在源域上进行了诊断任务的验证。**模型的训练过程中，通过迁移学习技术对源域模型进行优化，分析了模型在源域数据上的表现，并进行了性能评估，为后续的目标域迁移学习奠定了基础。

问题 3 迁移诊断。为了应对目标域（即实际运营数据）的数据稀缺和标签不平衡问题，本文**采用了迁移学习方法，将源域训练得到的诊断模型迁移至目标域。通过特征迁移和参数微调，优化了目标域的故障诊断模型，成功提升了目标域数据的分类准确率。**通过迁移学习，不仅解决了目标域样本不足的问题，还显著提高了目标域在复杂环境下的故障诊断能力。

问题 4 迁移诊断的可解释性。考虑到迁移学习模型的“黑箱”问题，本文进一步研究了模型的可解释性。**通过事前可解释性分析、迁移过程可解释性分析和事后可解释性分析，揭示了源域和目标域特征之间的迁移路径。采用 SHAP 值分析、t-SNE 可视化和 PCA 降维等技术，对迁移学习过程中的特征变化及其对故障诊断结果的影响进**

行了深入分析。通过可解释性分析，不仅提高了模型的透明度，还增强了使用者对模型结果的理解与信任。

1.1 问题 1：数据分析与故障特征提取

1.1.1 问题分析

在这个任务中，我们的目标是从提供的源域数据（轴承试验台架振动数据）中筛选部分数据，并结合轴承的故障机理，提取有效的故障特征。我们需要根据轴承故障的类型和特征，选取合适的方法进行特征提取，为后续的故障诊断任务提供有力支持。轴承故障的类型包括外圈故障（OR）、内圈故障（IR）、滚动体故障（B），以及正常状态（N）。通过数据分析，我们需要从时域、频域、时频域等多个维度进行特征提取，进而有效描述这些故障状态。

1.1.2 具体实现思路

数据预处理：

- **去噪**：由于数据中可能包含噪声或其他干扰因素，我们需要对数据进行去噪处理（例如使用小波变换、滤波器等）。
- **归一化/标准化**：为了提高模型的稳定性，通常需要对数据进行归一化或标准化处理。
- **数据分割**：将源域数据划分为训练集和测试集，确保数据集的代表性。

特征提取：

- **时域特征**：提取信号的均值、方差、峰度、峭度、偏度等统计特征。这些特征能够捕捉到振动信号的基本波动性。
- **频域特征**：通过傅里叶变换将信号转化为频域，提取功率谱密度、频率特征（如 BPFO、BPFI、BSF）。这些频率特征可以用来表示轴承的故障频率。
- **时频域特征**：通过短时傅里叶变换（STFT）或小波变换（CWT）对信号进行时频分析，提取时变的频率特征。
- **图像特征**：将时域或频域信号转换为图像形式，使用卷积神经网络（CNN）提取高级特征。

特征选择与降维：

- 使用 PCA（主成分分析）或 LDA（线性判别分析）对特征进行降维，减少冗余信息，确保最终特征集包含有用的、能够区分故障状态的特征。

数据集构建：

- 从源域数据中选择有代表性的数据，构建训练集和测试集，并保证每类故障数据的比例。

1.1.3 具体实现步骤与代码

1.数据预处理：

读取数据并对其进行标准化和去噪处理。

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from sklearn.preprocessing import StandardScaler
import scipy.io as sio

# 读取数据（.mat 文件）
data = sio.loadmat('B007_0.mat')

# 假设数据包括 DE, FE, BA, RPM 等信号
DE = data['X118_DE_time'][0]
FE = data['X118_FE_time'][0]
BA = data['X118_BA_time'][0]
RPM = data['X118RPM'][0]

# 去噪处理（示例：使用带通滤波器）
fs = 12000  # 采样频率
lowcut = 100.0
highcut = 3000.0

def bandpass_filter(data, lowcut, highcut, fs, order=4):
    nyquist = 0.5 * fs
    low = lowcut / nyquist
    high = highcut / nyquist
    b, a = signal.butter(order, [low, high], btype='band')
    return signal.filtfilt(b, a, data)

DE_filtered = bandpass_filter(DE, lowcut, highcut, fs)
FE_filtered = bandpass_filter(FE, lowcut, highcut, fs)
BA_filtered = bandpass_filter(BA, lowcut, highcut, fs)
```

```
# 数据标准化
scaler = StandardScaler()
DE_scaled = scaler.fit_transform(DE_filtered.reshape(-1, 1))
FE_scaled = scaler.fit_transform(FE_filtered.reshape(-1, 1))
BA_scaled = scaler.fit_transform(BA_filtered.reshape(-1, 1))
```

2.特征提取：

提取时域、频域、时频域等特征。

```
from scipy.fft import fft
from scipy.signal import spectrogram

# 时域特征：均值、标准差、偏度、峭度等
def time_domain_features(data):
    mean = np.mean(data)
    std = np.std(data)
    skewness = np.mean((data - mean)**3) / std**3
    kurtosis = np.mean((data - mean)**4) / std**4 - 3
    return mean, std, skewness, kurtosis

DE_mean, DE_std, DE_skew, DE_kurt = time_domain_features(DE_scaled)

# 频域特征：频谱、功率谱密度（PSD）、特征频率（BPFO, BPFI, BSF）
def frequency_domain_features(data, fs):
    N = len(data)
    f = np.fft.fftfreq(N, 1/fs)
    fft_vals = fft(data)
    power = np.abs(fft_vals[:N//2])**2
    freq = f[:N//2]
    return freq, power

DE_freq, DE_power = frequency_domain_features(DE_scaled, fs)

# 时频域特征：短时傅里叶变换（STFT）或小波变换
def time_frequency_features(data, fs):
    f, t, Sxx = spectrogram(data, fs)
    return f, t, Sxx

DE_f, DE_t, DE_Sxx = time_frequency_features(DE_scaled, fs)

# 提取其它故障频率（例如 BPFO, BPFI 等）
# 以外圈故障 BPFO 为例：BPFO = f_r * N_d / 2 * (1 - d/D)
def calculate_BPFO(f_r, N_d, d, D):
```

```
    return f_r * N_d / 2 * (1 - d / D)

BPFO = calculate_BPFO(600, 9, 0.3126, 1.537)  # 示例数据
```

3.特征选择与降维：

使用 PCA 或 LDA 进行特征选择。

```
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA

# 合并特征
features = np.array([DE_mean, DE_std, DE_skew, DE_kurt]).reshape(1, -1)

# PCA 降维
pca = PCA(n_components=2)
features_pca = pca.fit_transform(features)

# LDA 降维（需要标签）
labels = np.array([0])  # 假设标签是 0，表示正常
lda = LDA(n_components=2)
features_lda = lda.fit_transform(features, labels)
```

4.构建数据集：

结合轴承故障机理，选择合适的源域数据，构建训练集和测试集。

```
# 划分训练集与测试集
from sklearn.model_selection import train_test_split

X = np.array([features_pca])  # 特征向量
y = np.array([0])  # 假设标签是 0（正常）

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

5.总结：

- **数据预处理：**对数据进行标准化和去噪处理。
- **特征提取：**提取时域、频域、时频域等特征，分析轴承故障的关键特征。
- **特征选择与降维：**使用 PCA 或 LDA 进行特征降维。
- **数据集构建：**通过划分训练集和测试集，为后续的模型训练提供数据支持。

1.2 问题 2：源域故障诊断

1.2.1 问题分析

在问题二中，我们的目标是基于任务一中提取的特征，构建并训练一个故障诊断模型，以完成源域数据（轴承试验台架数据）的故障诊断任务。源域数据已通过特征提取方法（时域、频域、时频域等）得到表示，因此接下来我们需要选择合适的机器学习或深度学习模型，进行训练、评估并实现故障分类。本问题的关键挑战是如何设计一个高效的诊断模型，并有效地从源域数据中提取出轴承故障的特征，以实现高精度的故障分类。源域数据集中的标签已知（包含外圈故障、内圈故障、滚动体故障和正常状态），因此可以采用有监督学习方法。

1.2.2 具体实现思路

数据准备：

- 使用任务一中提取的特征集（如时域特征、频域特征等）。
- 将源域数据划分为训练集和测试集。

选择合适的模型：

- 常见的故障诊断模型包括支持向量机（SVM）、决策树（DT）、随机森林（RF）、K 近邻（KNN）、神经网络（ANN）、卷积神经网络（CNN）等。具体选择哪种模型，取决于数据的特性和任务要求。
- 本示例中，我们选择使用随机森林（RF）模型，它适用于高维特征数据，并且易于实现和解释。

训练模型：

- 使用训练集数据训练诊断模型。
- 调整模型的超参数（如树的个数、最大深度等），以获得最佳性能。

评估模型：

- 使用测试集对模型进行评估，计算准确率、精确率、召回率、F1 值等评估指标。
- 可通过混淆矩阵可视化模型的分类效果。

诊断过程：

- 使用训练好的模型对目标域（列车轴承）数据进行诊断。
- 评估模型的迁移效果，并通过性能评估反馈模型的优劣。

1.2.3 具体实现步骤与代码

1.数据准备：

假设在任务一中我们已经提取了源域数据的特征，并且将其标准化处理。现在我们将这些特征和对应的标签（故障类型）准备好，划分训练集和测试集。

```
from sklearn.model_selection import train_test_split
import numpy as np

# 假设我们已经从任务一中提取了特征
# features 是一个形状为 (n_samples, n_features) 的矩阵，表示每个样本的特征
# labels 是一个形状为 (n_samples,) 的向量，表示每个样本的故障类型标签（0: 正常, 1: 外圈故障, 2: 内圈故障, 3: 滚动体故障）
X = np.array([DE_mean, DE_std, DE_skew, DE_kurt])  # 示例特征（你可以使用更多的特征）
y = np.array([0, 1, 2, 3])  # 示例标签（故障类型）

# 划分训练集和测试集（80%训练，20%测试）
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

2.选择并训练模型：

选择**随机森林（Random Forest）**模型进行训练。随机森林是集成学习的一种方法，通过构建多棵决策树来进行分类。

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# 创建随机森林分类器
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# 训练模型
rf_model.fit(X_train, y_train)

# 预测测试集
y_pred = rf_model.predict(X_test)

# 评估模型
```

```
print("Classification Report:\n", classification_report(y_test, y_pred))
```

3.评估模型性能：

计算准确率、精确率、召回率、F1 值等评估指标，并可视化混淆矩阵。

```
conf_matrix = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', xticklabels=['Normal', 'Outer Race Fault',
'Inner Race Fault', 'Rolling Element Fault'], yticklabels=['Normal', 'Outer Race Fault', 'Inner Race Fault',
'Rolling Element Fault'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

4.超参数调整（可选）：

使用交叉验证（cross-validation）进行超参数调整，以进一步提高模型的性能。

```
from sklearn.model_selection import GridSearchCV

# 定义超参数搜索范围
param_grid = {
    'n_estimators': [50, 100, 200],   # 树的数量
    'max_depth': [None, 10, 20, 30],   # 树的最大深度
    'min_samples_split': [2, 5, 10],   # 最小分裂样本数
    'min_samples_leaf': [1, 2, 4]        # 最小叶子节点样本数
}

# 网格搜索进行超参数调整
grid_search = GridSearchCV(estimator=rf_model, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)

# 训练模型
grid_search.fit(X_train, y_train)

# 输出最佳参数
print("Best Parameters: ", grid_search.best_params_)

# 使用最佳参数训练模型
best_rf_model = grid_search.best_estimator_

# 预测测试集
y_pred_best = best_rf_model.predict(X_test)
```

```
# 评估最佳模型
print("Best Classification Report:\n", classification_report(y_test, y_pred_best))
```

5.诊断过程：训练好的模型可以用于目标域数据的诊断任务，但由于目标域数据标签未知，直接在目标域上进行测试并评估模型的迁移效果。

```
# 假设目标域数据已准备好，并提取了相同的特征
# X_target 是目标域数据的特征（与源域数据特征相同）
X_target = np.array([target_DE_mean, target_DE_std, target_DE_skew, target_DE_kurt])

# 使用训练好的模型进行预测
y_target_pred = rf_model.predict(X_target)

# 输出诊断结果
print("Target Domain Predictions:", y_target_pred)
```

6.总结：

- **数据准备**：从任务一中提取的特征作为输入，标签作为输出，将源域数据划分为训练集和测试集。
- **选择模型**：选择随机森林作为诊断模型，并使用训练集进行训练。
- **模型评估**：通过准确率、精确率、召回率、F1 值等评估指标对模型进行评估，混淆矩阵可视化帮助理解模型分类效果。
- **超参数调整**：使用网格搜索方法对模型的超参数进行优化，提高模型性能。
- **迁移诊断**：在目标域数据上进行诊断，评估模型的迁移效果。

1.3 问题 3：迁移诊断

1.3.1 问题分析

在问题三中，我们的目标是通过迁移学习技术将源域（轴承试验台架数据）的故障诊断知识迁移到目标域（列车轴承数据）。由于目标域数据的标签未知，我们需要设计一种迁移学习方法，使得训练好的模型可以从源域数据的知识中“迁移”到目标域数据，实现准确的故障诊断。

迁移学习的核心思想是通过利用源域（已标注的数据）获得的知识，帮助目标域（无标签数据）实现更好的学习效果。由于源域和目标域的数据特性（如工况、采样

频率等）可能存在差异，因此迁移学习的任务是通过某种方式减少源域和目标域之间的分布差异，提升目标域数据的诊断能力。

1.3.2 具体实现思路

源域与目标域数据的分析：

- **源域数据**：来自试验台架的数据，已标注。
- **目标域数据**：来自实际列车的数据，未标注。需要进行迁移学习以利用源域的知识。

迁移学习方法的选择：

- 本任务中，我们选择**基于特征的迁移学习方法**，即通过对源域和目标域数据进行特征映射，使其在相同的特征空间内，从而减少源域与目标域之间的分布差异。
- 迁移学习可以采用**对抗训练**的方法，或使用**最大均值差异（MMD）**来对源域和目标域数据进行对齐。

迁移学习的实现步骤：

- **特征映射**：通过对源域和目标域数据的特征空间进行映射，将两者的分布尽量对齐。
- **模型微调**：使用源域数据训练的模型（如随机森林、神经网络等），并对目标域数据进行微调，使其适应目标域的特征。
- **迁移诊断**：应用微调后的模型对目标域的故障数据进行诊断，得到目标域数据的分类标签。

迁移学习的可解释性：

- 为了提高模型的透明性和可解释性，可以通过分析迁移学习过程中的特征变化，揭示源域与目标域的共性和差异。

1.3.3 具体实现步骤与代码

1.特征映射与对抗训练：

使用**对抗训练**（如生成对抗网络 GAN）或**最大均值差异（MMD）对源域和目标域的数据进行对齐。这里我们使用 MMD** 方法进行特征映射。

2.基于最大均值差异（MMD）的方法：

最大均值差异（MMD）是一种用于衡量源域和目标域数据分布差异的度量，目标是最小化源域和目标域之间的 MMD 值，从而将两者的分布对齐。

```
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.metrics import classification_report
from sklearn.preprocessing import StandardScaler

# MMD 计算方法
def calculate_mmd(source_features, target_features, kernel='rbf', gamma=1.0):
    # Source domain and target domain features are assumed to be torch tensors
    # MMD with RBF kernel
    n_source = source_features.size(0)
    n_target = target_features.size(0)

    XX = torch.matmul(source_features, source_features.T)
    YY = torch.matmul(target_features, target_features.T)
    XY = torch.matmul(source_features, target_features.T)

    if kernel == 'rbf':
        XX = torch.exp(-gamma * (XX - torch.diagonal(XX)).mean())
        YY = torch.exp(-gamma * (YY - torch.diagonal(YY)).mean())
        XY = torch.exp(-gamma * XY)

    mmd_loss = XX.mean() + YY.mean() - 2 * XY.mean()
    return mmd_loss

# 用于迁移学习的训练过程
class MMD_Net(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(MMD_Net, self).__init__()
        self.fc = nn.Linear(input_dim, output_dim)

    def forward(self, x):
        return self.fc(x)

# 假设源域和目标域的特征已准备好
X_source = np.random.randn(100, 10)  # 源域特征
X_target = np.random.randn(100, 10)  # 目标域特征

scaler = StandardScaler()
X_source = scaler.fit_transform(X_source)
```

```
X_target = scaler.transform(X_target)

X_source = torch.tensor(X_source, dtype=torch.float32)
X_target = torch.tensor(X_target, dtype=torch.float32)

# 初始化模型
model = MMD_Net(input_dim=10, output_dim=4)  # 4 类故障
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

# 训练过程：源域数据微调
num_epochs = 50
for epoch in range(num_epochs):
    model.train()

    optimizer.zero_grad()

    # 源域数据的输出
    y_source = model(X_source)  # 假设源域标签已知
    loss = criterion(y_source, torch.tensor([0, 1, 2, 3]))  # 示例标签，实际使用时替换为源域标签

    # MMD loss
    mmd_loss = calculate_mmd(X_source, X_target)

    total_loss = loss + 0.1 * mmd_loss  # 0.1 为 MMD 损失的权重

    total_loss.backward()
    optimizer.step()

    if epoch % 10 == 0:
        print(f"Epoch [{epoch}/{num_epochs}], Loss: {total_loss.item():.4f}, MMD Loss:
{mmd_loss.item():.4f}")

# 目标域数据迁移诊断
model.eval()
with torch.no_grad():
    y_target_pred = model(X_target)
    _, y_target_pred_labels = torch.max(y_target_pred, 1)
    print("Target Domain Prediction:", y_target_pred_labels)
```

3.基于深度学习模型的微调（Fine-tuning）：

采用源域训练好的深度学习模型（例如卷积神经网络或全连接神经网络）在目标域上进行微调。通过冻结源域训练的前几层（保持低级特征不变），仅对后续层进行训练，以适应目标域数据的特性。

```
from torch import nn
import torch.optim as optim

# 假设你已经使用源域数据训练了一个简单的神经网络（如 MLP）
class SimpleMLP(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(SimpleMLP, self).__init__()
        self.fc1 = nn.Linear(input_dim, 64)
        self.fc2 = nn.Linear(64, 32)
        self.fc3 = nn.Linear(32, output_dim)

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

# 使用源域训练的数据训练简单神经网络
source_model = SimpleMLP(input_dim=10, output_dim=4)
optimizer = optim.Adam(source_model.parameters(), lr=0.001)
criterion = nn.CrossEntropyLoss()

# 假设你有源域数据和目标域数据
X_source = torch.tensor(X_source, dtype=torch.float32)
y_source = torch.tensor([0, 1, 2, 3])  # 示例标签，实际替换为真实的源域标签

# 训练源域模型
source_model.train()
for epoch in range(100):
    optimizer.zero_grad()
    y_pred = source_model(X_source)
    loss = criterion(y_pred, y_source)
    loss.backward()
    optimizer.step()

    if epoch % 10 == 0:
        print(f'Epoch [{epoch}/100], Loss: {loss.item():.4f}')

# 微调模型：冻结前几层，只更新后几层
for param in source_model.fc1.parameters():
```

```
        param.requires_grad = False
for param in source_model.fc2.parameters():
        param.requires_grad = False

# 微调目标域数据
X_target = torch.tensor(X_target, dtype=torch.float32)
y_target_pred = source_model(X_target)  # 在目标域数据上进行预测
_, y_target_pred_labels = torch.max(y_target_pred, 1)
print("Target Domain Prediction after Fine-tuning:", y_target_pred_labels)
```

1.4 问题 4：迁移诊断的可解释性

1.4.1 问题分析

在机器学习和深度学习中，模型的可解释性是一个关键问题。特别是在迁移学习中，我们不仅希望模型能够有效地完成目标任务（例如故障诊断），还希望能够理解迁移过程中的决策依据，如何将源域的知识有效地迁移到目标域。理解模型如何做出决策以及迁移的过程，能够提高用户对模型的信任度，并有助于模型的优化和改进。

在本任务中，我们的目标是分析迁移诊断模型的可解释性。特别是要分析迁移学习中的**事前可解释性**、**迁移过程可解释性**和**事后可解释性**。通过可解释性分析，可以揭示源域和目标域之间的关系，理解模型如何迁移以及在迁移过程中如何做出诊断决策。

1.4.2 具体实现思路

事前可解释性：

- 事前可解释性指的是模型的结构和决策逻辑本身是透明的，能够从模型的输入特征和输出结果直接理解模型的行为。为了提高事前可解释性，我们可以选择一些本身具有良好可解释性的模型，如**决策树**、**随机森林**等。这些模型可以直观地展示特征重要性和决策过程。

迁移过程可解释性：

- 迁移过程可解释性指的是在迁移学习过程中，如何将源域的知识迁移到目标域，并揭示迁移的路径和方式。常用的方法包括**特征可视化**和**迁移学习的相关性分析**，比如使用 **t-SNE** 或 **PCA** 来可视化源域和目标域的特征空间的变化，从而直观地观察迁移过程中的特征对齐。

事后可解释性：

- 事后可解释性指的是对模型的预测结果进行分析，解释模型是如何根据输入数据做出决策的。常用的技术包括**特征重要性分析**、**LIME（局部可解释模型-agnostic 解释）**和 SHAP（Shapley 值）**等，这些方法可以帮助我们理解特征对于决策的贡献。

1.4.3 具体实现步骤与代码

1. 事前可解释性：特征重要性分析（基于随机森林）

通过随机森林模型来分析各个特征对于预测结果的重要性。随机森林能够提供每个特征的重要性评分，帮助我们理解哪些特征对诊断结果最为关键。

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split

# 生成模拟数据（你可以替换成自己的数据集）
X, y = make_classification(n_samples=1000, n_features=10, n_informative=5, n_classes=4,
random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 创建随机森林模型
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# 训练模型
rf_model.fit(X_train, y_train)

# 特征重要性
feature_importances = rf_model.feature_importances_

# 可视化特征重要性
plt.figure(figsize=(10, 6))
plt.bar(range(X.shape[1]), feature_importances)
plt.xlabel('Feature Index')
plt.ylabel('Feature Importance')
plt.title('Feature Importance Analysis using Random Forest')
plt.show()
```

通过上述代码，我们可以查看每个特征的重要性，进而理解哪些特征对故障诊断最为关键。对于迁移诊断，我们可以分析源域和目标域特征在迁移过程中的重要性变化，来解释迁移的过程。

2. 迁移过程可解释性：使用 t-SNE 可视化源域与目标域特征

为了分析源域和目标域之间的差异，可以使用 **t-SNE**（t-distributed Stochastic Neighbor Embedding）方法进行降维并可视化源域和目标域的数据分布。

```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# 假设我们已经有源域和目标域的特征
# X_source 和 X_target 分别是源域和目标域的特征
# 你可以使用从任务一或任务三中提取的特征数据
X_source = np.random.randn(100, 10)  # 模拟数据
X_target = np.random.randn(100, 10)  # 模拟数据

# 使用 t-SNE 进行降维
tsne = TSNE(n_components=2, random_state=42)
X_source_tsne = tsne.fit_transform(X_source)
X_target_tsne = tsne.fit_transform(X_target)

# 可视化源域和目标域特征
plt.figure(figsize=(10, 6))
plt.scatter(X_source_tsne[:, 0], X_source_tsne[:, 1], label="Source Domain", alpha=0.6)
plt.scatter(X_target_tsne[:, 0], X_target_tsne[:, 1], label="Target Domain", alpha=0.6)
plt.title('t-SNE Visualization of Source and Target Domain Features')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.legend()
plt.show()
```

通过 t-SNE 的可视化，我们可以清晰地观察到源域和目标域在特征空间中的分布差异。如果两者的分布较为相似，说明迁移的过程可能比较顺利；如果两者的分布差异较大，则说明需要更多的迁移策略来缩小这种差距。

3. 事后可解释性：SHAP 值分析

SHAP（Shapley Additive Explanations）值是一种可以解释模型输出的工具，尤其适用于复杂的黑箱模型（如随机森林和神经网络）。SHAP 值能够告诉我们每个特征对预测结果的贡献度。

```
import shap
import matplotlib.pyplot as plt

# 使用 SHAP 进行模型解释
# 假设我们已经训练了随机森林模型
explainer = shap.TreeExplainer(rf_model)
shap_values = explainer.shap_values(X_test)
```

```
# 绘制 SHAP 总结图
shap.summary_plot(shap_values, X_test, plot_type="bar")
```

4. 迁移过程可解释性：分析特征在源域和目标域的变化

为了更深入地理解迁移过程，可以通过**PCA（主成分分析）**来对源域和目标域数据进行降维，并分析它们在降维后的空间中的差异。

```
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# 使用 PCA 进行降维
pca = PCA(n_components=2)
X_source_pca = pca.fit_transform(X_source)
X_target_pca = pca.fit_transform(X_target)

# 可视化源域和目标域特征
plt.figure(figsize=(10, 6))
plt.scatter(X_source_pca[:, 0], X_source_pca[:, 1], label="Source Domain", alpha=0.6)
plt.scatter(X_target_pca[:, 0], X_target_pca[:, 1], label="Target Domain", alpha=0.6)
plt.title('PCA Visualization of Source and Target Domain Features')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend()
plt.show()
```

PCA 可视化帮助我们了解源域和目标域特征的变化，并可以揭示迁移学习过程中，特征空间是如何变化的，以及如何通过迁移学习对特征进行对齐。

5. 总结：

- **事前可解释性**：通过特征重要性分析，了解哪些特征在源域和目标域中对分类任务最为重要。
- **迁移过程可解释性**：通过 t-SNE 或 PCA 可视化源域和目标域特征的变化，揭示迁移学习的过程和特征对齐的效果。
- **事后可解释性**：使用 SHAP 值分析，理解模型在目标域数据上的决策过程，解释模型预测的依据。