華中科技大學

模式识别课程设计

基于 WT-EMD 去噪的管道泄漏检测与识别与 SVM 和随机森林分类效果的比较

院　　系　人工智能与自动化学院

专业班级　智能医学工程 2201

姓　　名　薛卜元

学　　号　U202215123

指导教师　刘文黎

2024 年 12 月 23 日

目录

1. 摘要

为进行管道泄漏检测研究，通过搭建管道实验平台，使用水听器、示波器收集各种泄露工况下的信号波形以供计算机分析，识别并定位泄漏工况。之后利用模式识别方法，对泄漏与非泄漏状态进行分类。在此报告中，使用的信号去噪方法为 WT-EMD 去噪，对工况的分类方法分别为 SVM 和随机森林。通过去噪之后的数据在分类时，在分类类别数较少（基础任务）时可以达到 90%以上的分类准确度；在分类类别较多的时候也可以达到 80%以上，且随机森林的分类效果优于 SVM。

2. 介绍

2.1 背景和目的

我国幅员辽阔、水资源总量巨大，但因人口基数庞大导致人均水资源占有量仅为全球人均占有量的 1/4，属于严重缺水国家。目前，管道运输以其占地少、铺设时间短、经济耗费低、可靠性高和连续性强等优点成为水资源输送中最主要的方式，然而，我国仅在 2022 年因管道泄漏浪费的水资源高达 819053.49 万立方米，占全国城市供水总量的 12.89%。2022 年，住建部以及发改委在 2022 年联合发布了一份通知，要求到 2025 年将城市公共供水管网的泄漏率降至 9%，目前距离这一目标还有一定的差距。对于城市给水管道运维而言，如何及时且有效地检测出给水管道的微小泄漏成为亟需解决的研究问题。

为进行管道泄漏检测研究，搭建管道实验平台。该平台由实验管道、抽水泵机、水听器和示波器组成。其中，实验管道为硬聚氯乙烯（PVC）管，内径 250 mm，可承受 0~2 MPa 的大气压强。分别在管道两端开设了由球形阀控制的进水出水口，并在管道实验区段开设最大孔径为 10 mm 的可调节式泄漏孔。实验采用的两个抽水泵机均为凌霄 CMI3 系列不锈钢卧式多级离心泵，最大运行环境温度 50°C，最大运行压力 8 MPa，满足给水管道模拟实验要求。实验数据通过与水听器相连的 TBS 1102X 数字示波器显示，并采用触发采集传输至 LSWL-1OSC2.0-2 软件保存在电脑硬盘中。

2.2 设计任务

本次任务的基础目标就是利用供水管道的运行数据，应用 SVM 和随机森林的模式识别方法，对泄漏与非泄漏状态进行分类，识别并定位泄漏工况。

以下为任务的具体流程：

1. 使用小波变换和 EMD 对原始的信号数据进行去噪处理。

2. 对经过去噪处理之后的数据进行特征提取。所使用到的提取到的特征有：主频率、中心频率、峰度、均方根、峰值因子，另外实验过程中还提取了各个样本数据的频带宽度、香农熵、样本熵，但是发现各类样本之间的频带宽度、香农熵、样本熵无太大差别，担心可能影响到后续的 SVM 和随机森林分类，便将其剔除，不再提取这些特征。

3. 特征值均值图像绘制。生成并比较比较每种特征提取方法对工况区分的效果图。

4. 构建识别模型。本次报告使用了两种模型，SVM 和随机森林进行工况分类，以进行分类效果对比。

3. 研究方法

3.1 研究流程

首先对所有的实验数据进行去噪处理；之后对去噪后的数据进行特征提取，并绘制特征值均值图像；最后构建模式识别分类模型，对提取到的管道泄漏特征进行识别。

3.2 数据样本

本次实验所使用的数据样本为通过管道实验平台实验所得到的 16 组不同故障工况外加 1 组正常工况共 17 组的的信号数据，考虑到数据量大小只使用各个工况 70%的数据量。

3.3 实验环境

本次实验使用环境为 Python 3.11.5。

所使用的依赖库有：numpy 1.26.4、pandas 2.1.1、pyetnrp 1.0.0、matplotlib 3.8.0、scikit-learn 1.3.1、EMD-signal 1.6.4、PyWavelets 1.8.0、seaborn 0.12.2。

3.4 详细步骤

3.4.1 小波变换去噪

详细函数代码如下图所示：

Project - WT-EMD.py

```python
# 小波去噪函数
def wavelet_denoising(signal, wavelet='haar', level=4, threshold=0.05):
    # 小波变换
    coeffs = pywt.wavedec(signal, wavelet, level=level)
    # 阈值处理
    coeffs_thresholded = [pywt.threshold(c, threshold * np.max(c), mode='soft') for c in coeffs]
    # 小波重构
    denoised_signal = pywt.waverec(coeffs_thresholded, wavelet)

    return denoised_signal
```

小波变换可以将信号表示为一系列尺度不同的基函数的线性组合。这些基函数是小波函数，它们是通过平移和缩放从一个母小波（在此处为 haar）得到的。

连续小波变换将信号$x(t)$与小波函数$\psi(t)$进行卷积运算，生成一个小波变换系数：

$$W_x(a,b) = \int_{-\infty}^{\infty} x(t) \frac{1}{\sqrt{a}} \psi(\frac{t-b}{a})\, dt$$

其中，a是尺度因子，决定了小波的伸缩；b是平移因子，决定了小波的位置；$\psi(t)$是母小波函数。

小波去噪的关键步骤是对小波系数进行阈值处理。噪声通常表现为高频信号，而小波变换能够将高频噪声成分与低频信号分开。因此，通过对小波系数进行阈值化处理，能够去除高频噪声。软阈值处理：

$$\hat{c}_j = \begin{cases} sign(c_j)(|c_j| - \lambda), & |c_j| \geq \lambda \\ 0, & |c_j| < \lambda \end{cases}$$

软阈值方法不仅会将小于阈值的系数设为零，还会对大于阈值的系数进行缩小。软阈值处理通常比硬阈值方法（只有大于或等于阈值的系数保留，其他系数设为零）具有更好的去噪效果，因为它能更平滑地抑制噪声。

小波重构：小波重构指在阈值处理完成后，需要将处理后的系数重新组合成去噪后的信号。重构的过程实际上是小波变换的逆过程，使用阈值化后的系数来合成一个去噪信号。对于小波系数$\hat{c}_j$，使用以下重构公式得到去噪后的信号：

$$\hat{x}(t) = \sum_j \hat{c}_j \, \psi_j(t)$$

3.4.2 EMD 去噪

EMD 去噪的关键思想是从 EMD 分解得到的 IMFs 中去除噪声部分，保留有用的信号部分。通常，噪声主要集中在较高频的 IMFs 中，而低频部分包含的是信号的主要成分。

EMD 的过程将一个信号分解成若干个本征模态函数（IMFs）和一个剩余趋势项。首先，给定一个信号 x(t)，EMD 算法会找到信号中的局部极大值点和极小值点。通过局部极大值点和极小值点，分别通过样条插值法构造上包络线（连接所有局部极大值）和下包络线（连接所有局部极小值）。通过求取上包络线和下包络线的平均值，得到该信号的局部均值：

$$m(t) = \frac{upper\ envelope(t) + lower\ envelope(t)}{2}$$

然后，从原始信号中去除局部均值，得到残差信号：

$$h(t) = x(t) - m(t)$$

该残差信号$h(t)$被认为是一个新的信号，接下来继续进行上述的极值点确定和包络线构造过程，直到满足停止条件。

重复以上过程，直到得到一个本征模态函数（IMF）。IMF 的条件是它应该满足两个标准：在信号的所有局部极大值点和局部极小值点之间，信号的零交点数不超过 1；上包络线与下包络线的平均值与 IMF 的零均值要求相符。完成分解后，信号$x(t)$可以表示为一系列的 IMFs 加上一个趋势项：

$$x(t) = \sum_{i=1}^{n} IMF_i(t) + residual(t)$$

其中，$IMF_i(t)$是第i个本征模态函数，$residual(t)$是剩余的趋势项。

EMD 去噪方法的优点是能够自适应地处理信号的不同频率成分，并且对非平稳信号有较好的处理效果。它不依赖于预设的基函数，能够自动地识别信号的内在结构，从而有效去除噪声。

详细函数代码如下图所示：

其中 adaptive_threshold 函数可以基于标准差和能量自适应计算去噪阈值；k_std 为标准差倍数，决定去噪的强度；energy_ratio_threshold 为能量阈值，用于判断 IMF 是否为噪声。

Project - WT-EMD.py

```python
def adaptive_threshold(imf, k_std=0.2, energy_ratio_threshold=0.05):
    std_dev = np.std(imf) # 计算IMF的标准差
    energy = np.sum(imf ** 2) # 计算IMF的能量 使用方差来近似能量
    total_energy = np.sum(imf ** 2) # 计算该IMF的能量与所有IMF的能量的比值
    threshold_std = k_std * std_dev # 计算IMF的标准差阈值
    # 判断是否去除IMF
    # 如果IMF的能量比非常小且标准差小于阈值，则认为是噪声
    if energy / total_energy < energy_ratio_threshold and std_dev < threshold_std:
        return True # 被认为是噪声
    else:
        return False # 被认为是信号

# EMD去噪函数
def emd_denoising(signal, k_std=0.2, energy_ratio_threshold=0.05):
    emd = EMD() # 执行经验模态分解
    imfs = emd.emd(signal)
    # 对每个IMF使用自适应阈值判断是否去噪
    denoised_imfs = []
    for imf in imfs:
        if adaptive_threshold(imf, k_std, energy_ratio_threshold):
            denoised_imfs.append(np.zeros_like(imf)) # 如果是噪声IMF，设置为零
        else:
            denoised_imfs.append(imf) # 保留信号IMF
    denoised_signal = np.sum(denoised_imfs, axis=0) # 重构去噪后的信号

    return denoised_signal
```

3.4.3 计算去噪效果

将 WT-EMD 去噪后的信号与原始信号进行对比，并计算二者之间的信噪比 SNR、绝对误差 MAE 与均方误差 MSE。使用基于幅度的信噪比算法（Amplitude-based SNR）：

$$SNR(dB) = 10\log_{10}(\frac{A_{signal}^2}{A_{noise}^2})$$

其中，A_{signal}是信号的幅度，A_{noise}是噪声的幅度。

Project - WT-EMD.py

```python
def calculate_snr(original_signal, denoised_signal):
    noise = original_signal - denoised_signal
    signal_power = np.sum(original_signal**2)
    noise_power = np.sum(noise**2)
    snr = 10 * np.log10(signal_power / noise_power)
    return snr
```

计算 MAE：

$$MAE = |x_{true} - x_{pred}|$$

其中，前者为实际值，后者为预测值。

```
Project - WT-EMD.py
# 计算平均绝对误差(MAE)
def calculate_mae(original_signal, denoised_signal):
    return mean_absolute_error(original_signal, denoised_signal)
```

计算 MSE：

```
Project - WT-EMD.py
# 计算均方误差(MSE)
def calculate_mse(original_signal, denoised_signal):
    return mean_squared_error(original_signal, denoised_signal)
```

3.4.4 特征提取

在此次实验中，所提取的特征有：主频率 dominant frequency、中心频率 central frequency、峰度 kurtosis、均方根 RMS 与峰值因子 crest factor。在运行特征提取的代码时，每次输入的数据都是同一工况下的数据，并且同一个工况的提取出的特征保存在独立的 csv 表格中，方便之后训练集的生成。

计算主频率：

```
Project - Feature Extraction.py
def dominant_frequency(signal, sampling_rate=1250):
    fft_result = np.fft.fft(signal)
    freqs = np.fft.fftfreq(len(signal), 1 / sampling_rate) # 计算频率轴
    magnitude = np.abs(fft_result) # 获取频谱幅值
    dominant_freq_index = np.argmax(magnitude[:len(magnitude) // 2]) # 找到主频率的索引
    dominant_freq = np.abs(freqs[dominant_freq_index])
    return dominant_freq
```

计算均方根：

```
Project - Feature Extraction.py
def RMS(signal):
    rms_value = np.sqrt(np.mean(signal ** 2))
    return rms_value
```

均方根计算公式：

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}$$

其中，N为数据点的总数，x_i为数据集中的第i个数据值。

计算中心频率：

Project - Feature Extraction.py

```python
def central_frequency(signal, sampling_rate=1250):
    fft_result = np.fft.fft(signal) # 对信号进行傅里叶变换
    freqs = np.fft.fftfreq(len(signal), 1 / sampling_rate) # 计算频率轴
    magnitude = np.abs(fft_result) # 获取频谱幅值
    # 计算加权平均频率
    central_freq = np.sum(freqs[:len(freqs) // 2]
                          * magnitude[:len(magnitude) // 2]) / np.sum(magnitude[:len(magnitude) // 2])
    return central_freq
```

计算峰度：

Project - Feature Extraction.py

```python
def kurtosis(signal):
    mean_signal = np.mean(signal) # 计算信号的均值
    fourth_moment = np.mean((signal - mean_signal) ** 4) # 计算信号的四阶矩（中心化后）
    second_moment = np.mean((signal - mean_signal) ** 2) # 计算信号的二阶矩（中心化后）
    kurtosis = fourth_moment / (second_moment ** 2) # 峰度公式
    return kurtosis
```

峰度计算公式：

$$kurtosis = \frac{N(N-1)}{(N-2)(N-3)}\sum_{i=1}^{N}\left(\frac{x_i - \bar{x}}{s}\right)^4 - \frac{3(N-1)^2}{(N-2)(N-3)}$$

其中，N为数据点的总数，x_i为数据集中的第i个数据值，$\bar{x}$为样本的均值，s为样本的标准差。

计算峰值因子：

Project - Feature Extraction.py

```python
def crest_factor(signal):
    max_value = np.max(np.abs(signal)) # 计算信号的最大值和均方根
    rms_value = RMS(signal)
    crest_factor = max_value / rms_value # 计算峰值因子
    return crest_factor
```

3.4.5 特征值均值图像绘制与训练集生成

在此步骤中，将会计算不同工况的各个特征值的均值并将其绘制为折线图，以便于对比各个工况之间的不同。同时将会合并上一步中生成的不同工况的相互独立的 csv 文件，成为训练集以供 SVM 和随机森林使用。

3.4.6 支持向量机（SVM）分类

支持向量机 SVM 是一种监督学习算法，用于分类和回归问题，它的核心思想是找到一个最优的超平面来将数据分成不同的类别。在二维空间中，超平面就是一条线；在三维空间中，超平面是一个平面；在更高维空间中，超平面是一个高维空间中的平面。

对于一个线性可分的问题来说，一个超平面可以表示为：

$$\boldsymbol{w} \cdot \boldsymbol{x} + b = 0$$

其中，$\boldsymbol{w}$为超平面的法向量，b为偏置项。

对于每一个训练数据点，计算点到超平面的距离，类别为+1 的点位于超平面的正侧，类别为-1 的点位于超平面的负侧。为了让两类数据点之间的间隔最大化，需要确保所有数据点都位于超平面的一侧，同时距离超平面越远越好。间隔是两类支持向量之间的最小距离。支持向量是指那些位于距离超平面最近的点，它们决定了分割超平面的位置。该超平面两侧的平行超平面分别为：

$$\begin{cases} = +1 \\ \boldsymbol{w} \cdot \boldsymbol{x} + b = -1 \end{cases}$$

这两条平行超平面之间的间隔为：

$$\frac{2}{\|\boldsymbol{w}\|}$$

SVM 的目标便是最大化超平面之间的间隔$\frac{2}{\|\boldsymbol{w}\|}$。

为了确保每个数据点都被正确分类且位于相应的平行超平面的一侧，还需要满足每个数据点要么位于超平面的正侧要么位于超平面的负侧。即：

$$y_i(\boldsymbol{w} \cdot \boldsymbol{x} + b) \geq 1$$

$$\begin{cases} y_i = +1, & \textit{数据点位于超平面正侧} \\ y_i = -1, & \textit{数据点位于超平面负侧} \end{cases}$$

因此，SVM 的优化问题可以表示为：

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2$$

目标为最小化$\frac{1}{2}\|\boldsymbol{w}\|^2$，同时有$y_i(\boldsymbol{w} \cdot \boldsymbol{x} + b) \geq 1$。

3.4.7 随机森林分类

随机森林的核心是集成学习，它由多个决策树组成。每棵树都是通过对训练数据的随机采样和特征选择构建的。通过对这些决策树的结果进行投票或平均，得到最终的预测。

对于每棵决策树，随机森林从原始训练集$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$中通过有放回抽样生成一个新的训练集$D_{tree}$。这意味着每棵树的训练集可能会有重复的样本，且每棵树的训练集的样本数是与原训练集相同的。在构建每棵决策树时，不是考虑所有的特

征，而是从总特征集合中随机选择一个子集，称为“特征子集”。通常，选择的特征子集大小是$\sqrt{p}$或$\log_2 p$，其中p是特征的总数。

在分类问题中，对于一个新的样本$\boldsymbol{x}^*$，每棵树T_i都会给出一个类别预测$y_i = T_i(\boldsymbol{x}^*)$最终的预测结果是所有树的投票结果，即选择投票数最多的类别。

通过集成这些树的预测，随机森林能够有效避免过拟合，并提高模型的准确性。

3.5 数据分析方法

在对数据进行去噪之后，去噪信号与原信号之间的对比使用了计算 SNR、MAE 和 MSE 的方法来分析去噪效果。

对信号进行特征提取，提取了 5 个特征：主频率 dominant frequency、中心频率 central frequency、峰度 kurtosis、均方根 RMS 与峰值因子 crest factor，并生成不同工况下特征均值的拟合曲线图像。

最后的分类步骤中使用了混淆矩阵、准确率、精确率、召回率、F1 分数评估模型性能并对 SVM、随机森林两种方法进行了效果比较。

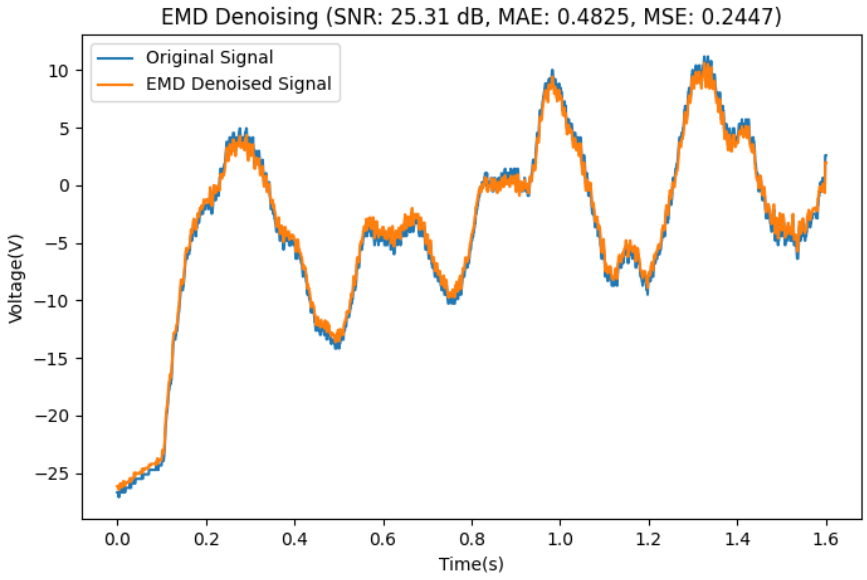
4. 结果与讨论

运行代码请参考文件夹中的 README.md 文件。

4.1 结果展示

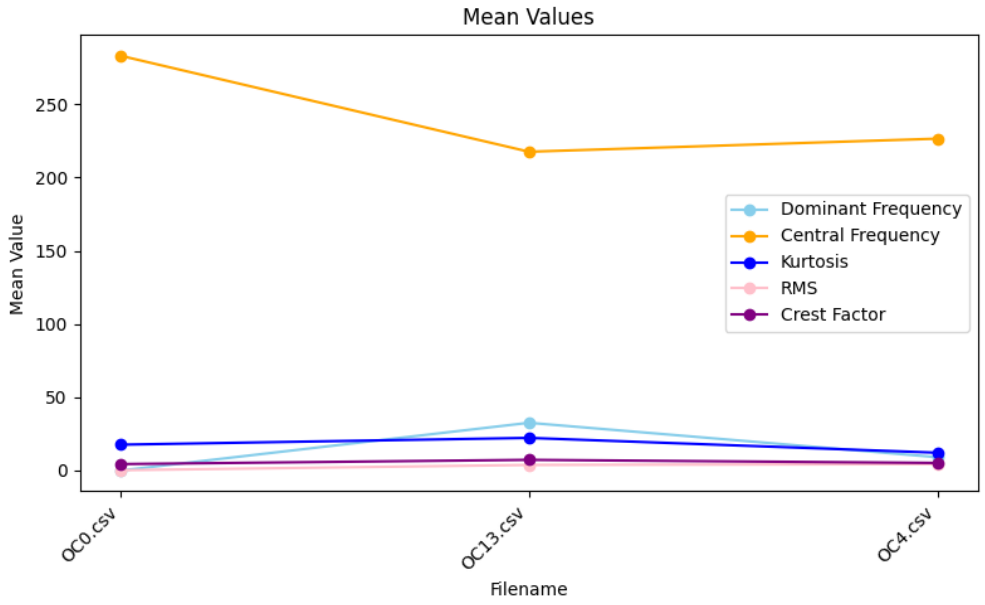
4.1.1 WD-EMD 去噪

运行程序之后生成对应的去噪信号文件，并将去噪信号与原信号之间的对比，计算 SNR、MAE 和 MSE。



4.1.2 特征提取和特征值均值图像绘制以及训练集生成

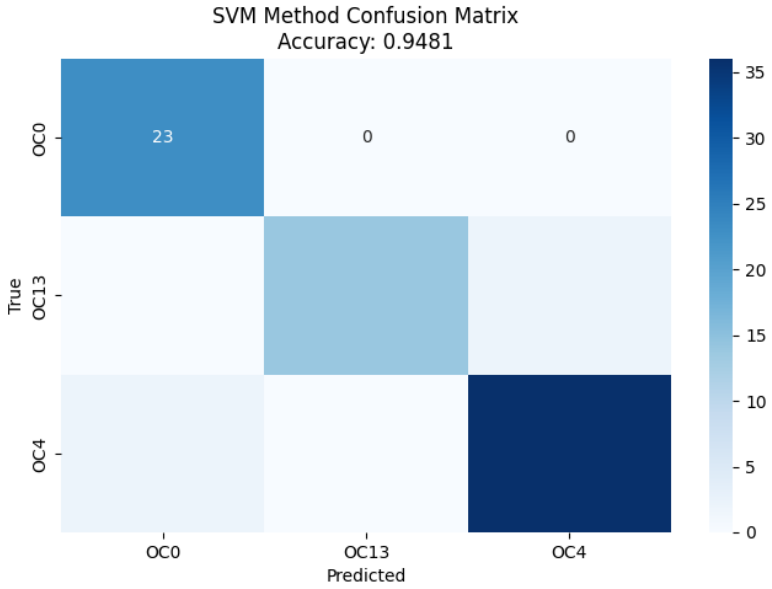
手动修改 Feature Extraction.py 文件中的文件夹路径并运行，可以得到所需要的工况的特征值文件。再运行 Graph & Merge.py 文件，即可得到特征值均值图像，并同时生成训练集以供分类模型使用。



4.1.3 SVM 以及随机森林分类结果

在正文限于格式和篇幅，只展示基础任务、进阶任务 1 和进阶任务 2 的相关图像，其余的进阶任务图像附于附录。

在基础任务，即分类工况 0、4、13 的任务条件下，SVM 方法和随机森林方法都可以很好的完成分类任务，且分类准确率均高于 90%。

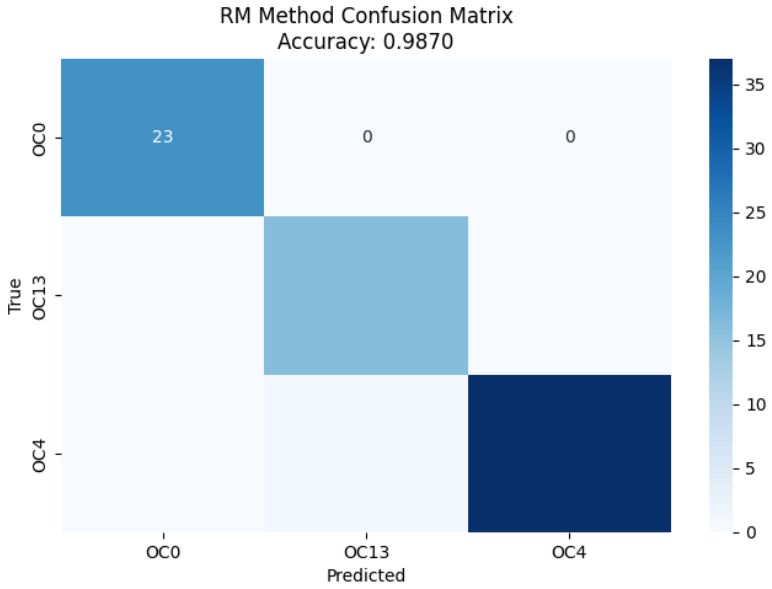


Project - SVM.py

```
Classification Report:
              precision    recall  f1-score   support

         OC0       0.92      1.00      0.96        23
        OC13       1.00      0.88      0.93        16
         OC4       0.95      0.95      0.95        38

    accuracy                           0.95        77
   macro avg       0.96      0.94      0.95        77
weighted avg       0.95      0.95      0.95        77
```



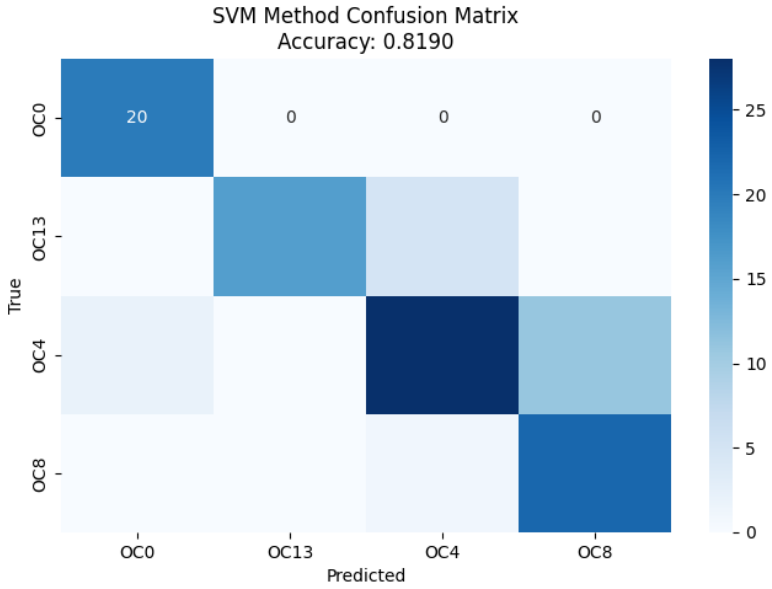
Project - Random Forest.py

```
Classification Report:
              precision    recall  f1-score   support

         OC0       1.00      1.00      1.00        23
        OC13       0.94      1.00      0.97        16
         OC4       1.00      0.97      0.99        38

    accuracy                           0.99        77
   macro avg       0.98      0.99      0.99        77
weighted avg       0.99      0.99      0.99        77
```

在进阶任务中：当需要分类的工况为 0、4、8、13 时，SVM 方法的准确率开始有明显下降，只有 82%左右，而随机森林的准确率仍然优秀。

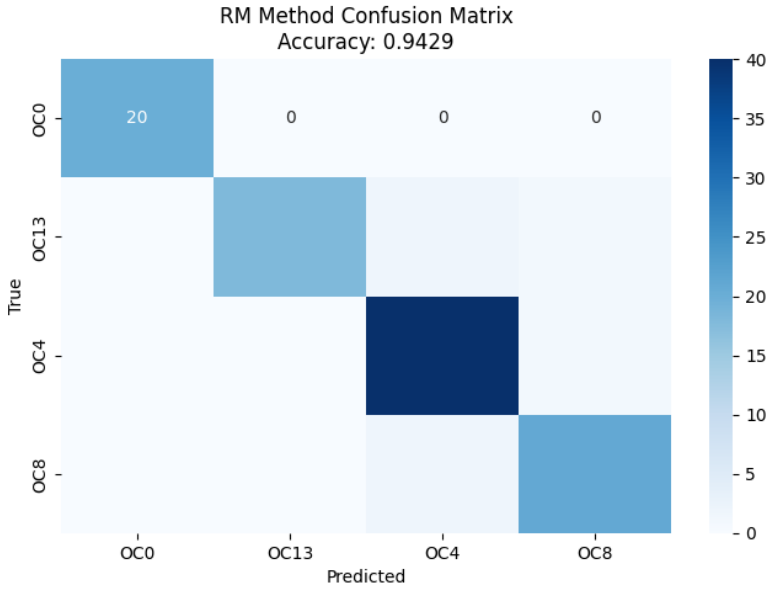


Project - SVM.py

```
Classification Report:
              precision    recall  f1-score   support

         OC0       0.91      1.00      0.95        20
        OC13       1.00      0.76      0.86        21
         OC4       0.82      0.68      0.75        41
         OC8       0.67      0.96      0.79        23

    accuracy                           0.82       105
   macro avg       0.85      0.85      0.84       105
weighted avg       0.84      0.82      0.82       105
```



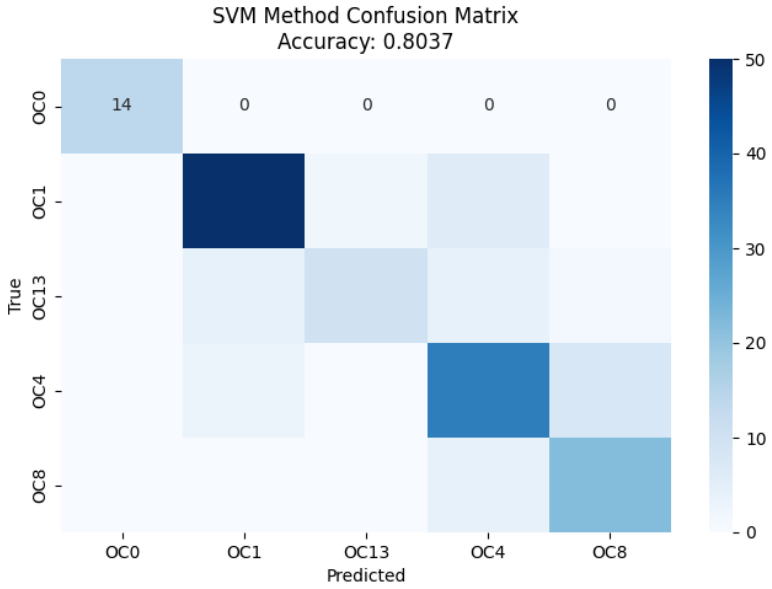
Project - Random Forest.py

```
Classification Report:
              precision    recall  f1-score   support

         OC0       1.00      1.00      1.00        20
        OC13       1.00      0.86      0.92        21
         OC4       0.91      0.98      0.94        41
         OC8       0.91      0.91      0.91        23

    accuracy                           0.94       105
   macro avg       0.96      0.94      0.94       105
weighted avg       0.95      0.94      0.94       105
```

当需要分类的工况为 0、1、4、8、13 时：SVM：80.37%，随机森林：96.32%。

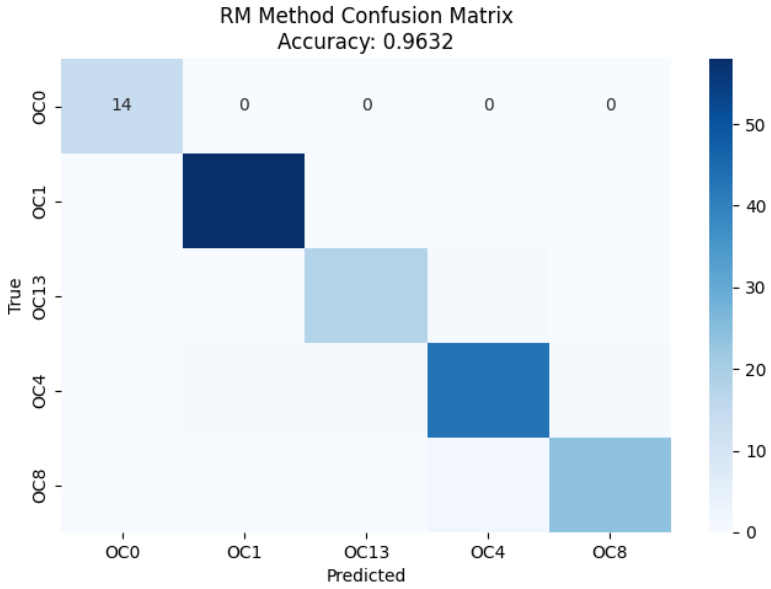


Project - SVM.py

```
Classification Report:
              precision    recall  f1-score   support

         OC0       1.00      1.00      1.00        14
         OC1       0.88      0.86      0.87        58
        OC13       0.83      0.53      0.65        19
         OC4       0.71      0.76      0.74        46
         OC8       0.71      0.85      0.77        26

    accuracy                           0.80       163
   macro avg       0.83      0.80      0.80       163
weighted avg       0.81      0.80      0.80       163
```



Project - Random Forest.py

```
Classification Report:
              precision    recall  f1-score   support

         OC0       1.00      1.00      1.00        14
         OC1       0.98      1.00      0.99        58
        OC13       0.95      0.95      0.95        19
         OC4       0.93      0.93      0.93        46
         OC8       0.96      0.92      0.94        26

    accuracy                           0.96       163
   macro avg       0.97      0.96      0.96       163
weighted avg       0.96      0.96      0.96       163
```

4.2 结果分析

当需要分类的类别较少时（基础任务三种工况），SVM 和随机森林方法都可以很好的完成分类任务，并且准确率、精确率、召回率、F1 分数都很好。但是当分类任务变得复杂时，SVM 的准确率下降较为严重（94.81%→81.90%→80.37%→79.89%），反观随机森林方法，其效果在各个分类任务中都很良好（98.70%→94.29%→96.32%→95.40%）。

发生这种情况的原因可能是数据本身较为复杂（主频率、中心频率、峰度、均方根与峰值因子 5 个特征），导致 SVM 在没有适当调参或核函数选择不当时难以捕捉到数据的模式。随机森林由于集成了多棵决策树，对大规模数据集也较为适应，而且受到数据噪声的影响也要更小。

5. 总结

此次实验在两种方法的分类准确率方面非常突出，特别是随机森林对于多分类问题的准确率非常优秀。相比起来，最初选用的 SVM 分类器在多分类问题上的准确率随着分类类别的增加而降低的速率很快。

在开始实验之前，对本次实验所使用的代码语言进行了一次抉择。MATLAB 非常擅长对数据的处理，而本次实验的基础便是良好的数据处理。但是我并不擅长 MATLAB 程序的编写，故而转向纯 Python 编写此次实验的代码。且 Python 在后续的模型训练方面更为得心应手。

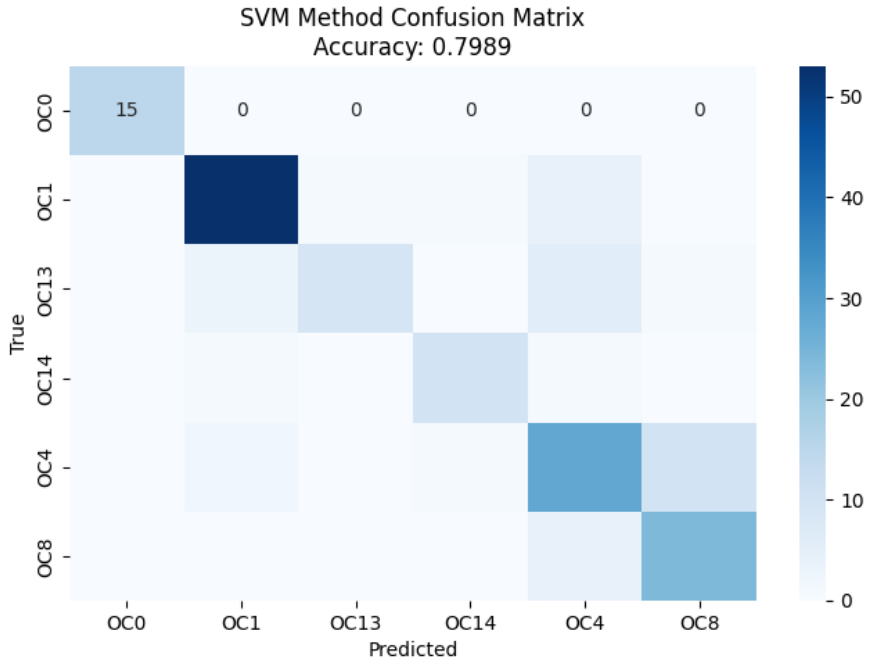
在此次实验的过程中，首先遇见的问题便是数据的初步处理。平均每种工况有大约 400 份信号数据，这么大量的数据需要整理、分类以及选用，所耗费的时间较多。在对信号进行去噪的过程中，WT 和 EMD 去噪的参数调整也遇见了困难，二者所需要调整的参数组合多样，而且某一组参数可能对这种波形的信号数据去噪效果好，但又对另一组波形的信号数据效果极差。一组相对最优的参数需要进行反复的测试才能够得到，而信号的去噪效果又与后续的模型分类效果息息相关，无法下定论在实验中所使用的这组去噪参数为绝对的最优参数。此次实验的初步优化方向除去完全新增新的去噪算法以替代旧算法之外，最简单的做法便是投入时间去细致调试去噪参数。

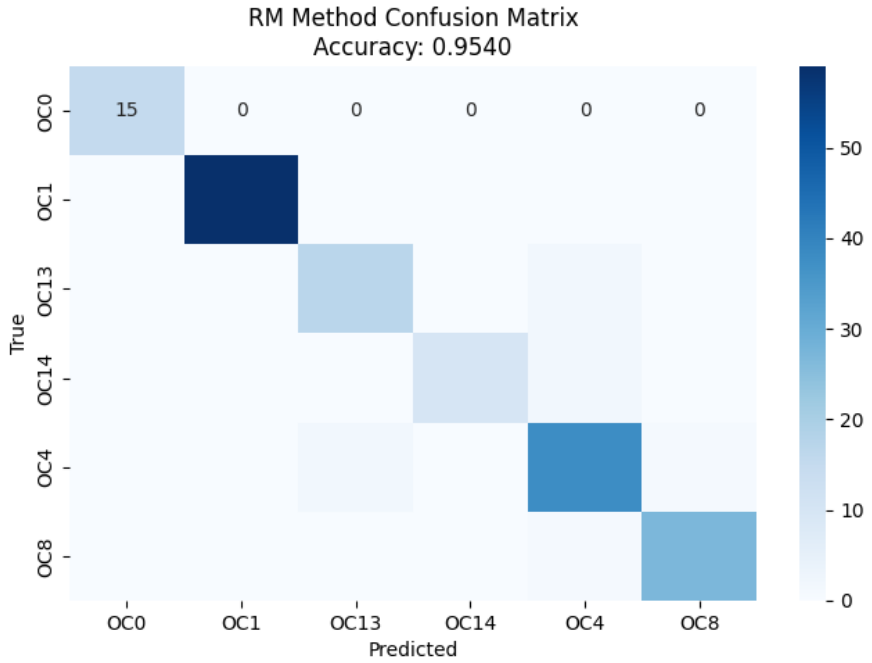
然后便是特征提取的部分。除去学习各种特征的数学原理之外，还需要通过编写代码来初步观察不同工况数据的各种特征是否有明显差距。要是差距并不明显，则不会选用于最后的训练集中。最后抛弃了频带宽度、香农熵、模糊熵和样本熵，选用主频率、中心频率、峰度、均方根与峰值因子 5 个特征。

6. 附录

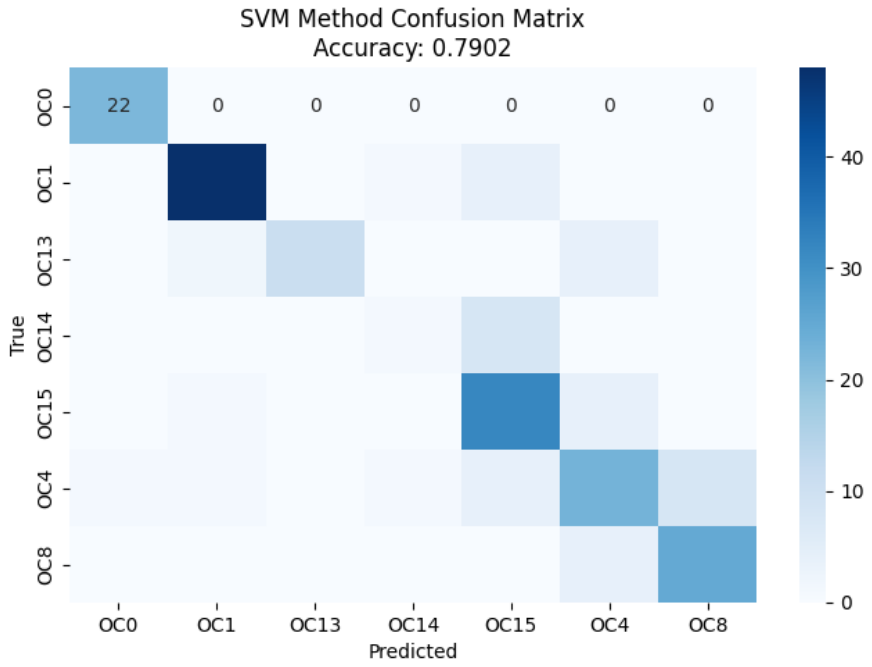
SVM 和随机森林在其他场景下的分类效果，附混淆矩阵与准确度图片于下。

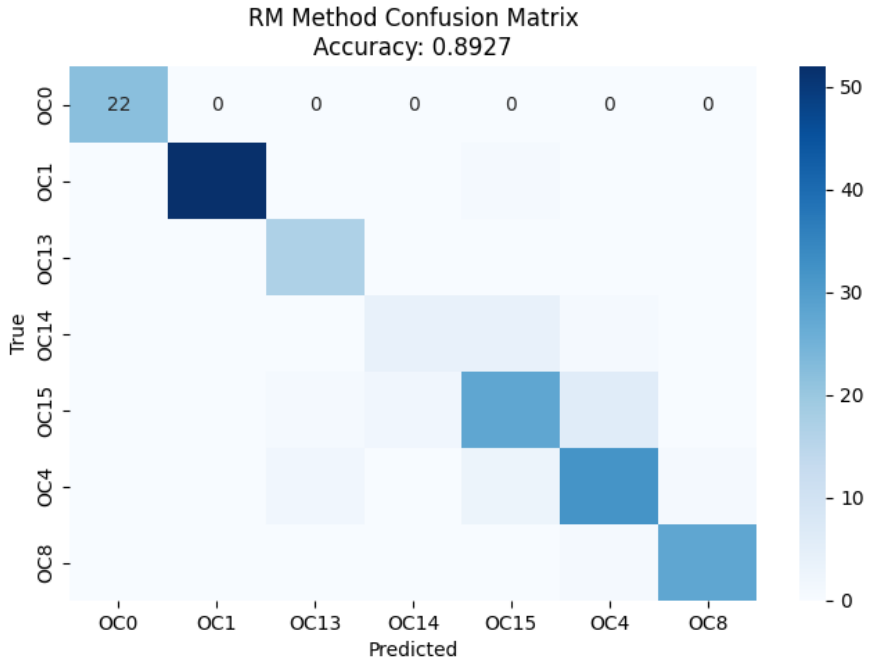
工况 0、1、4、8、13、14



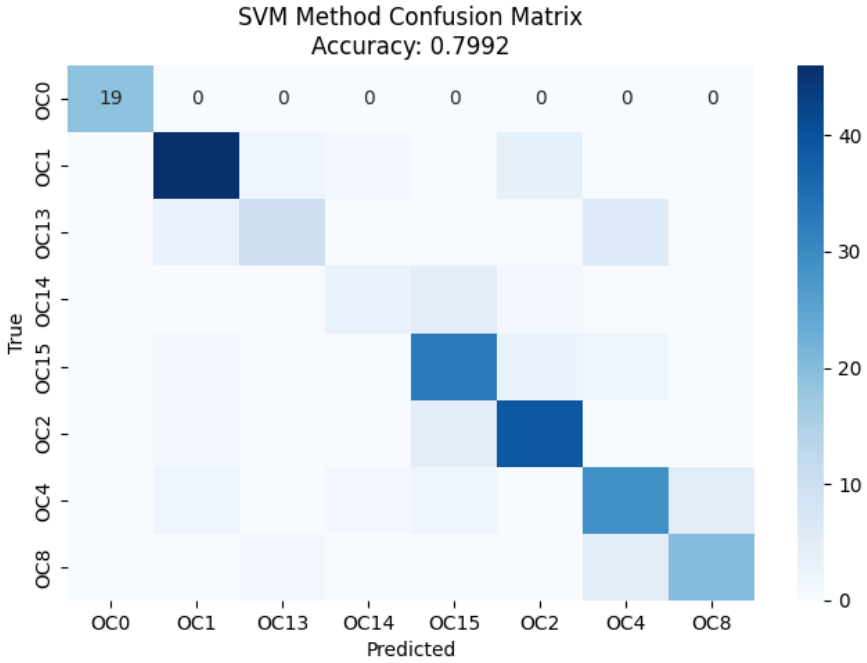


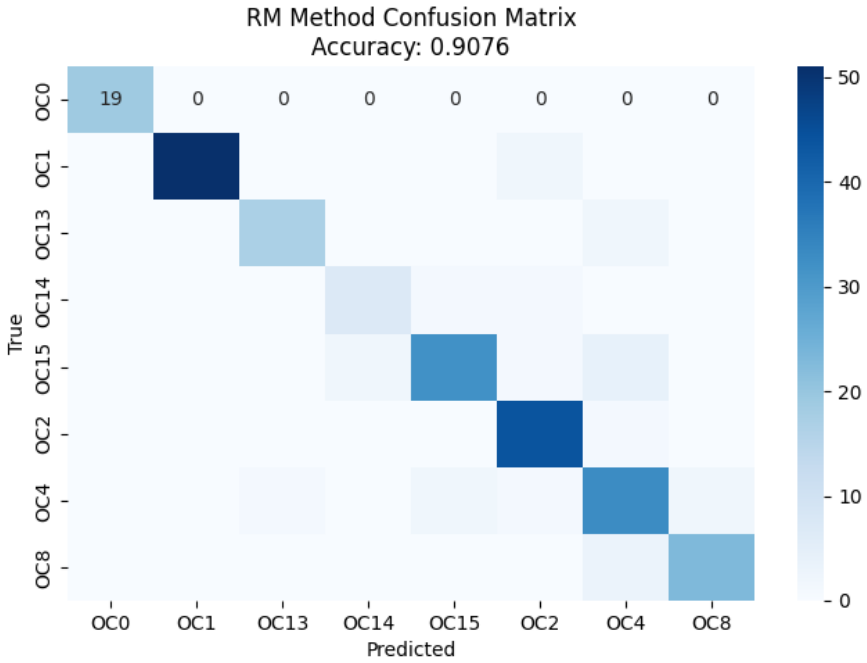
工况 0、1、4、8、13、14、15



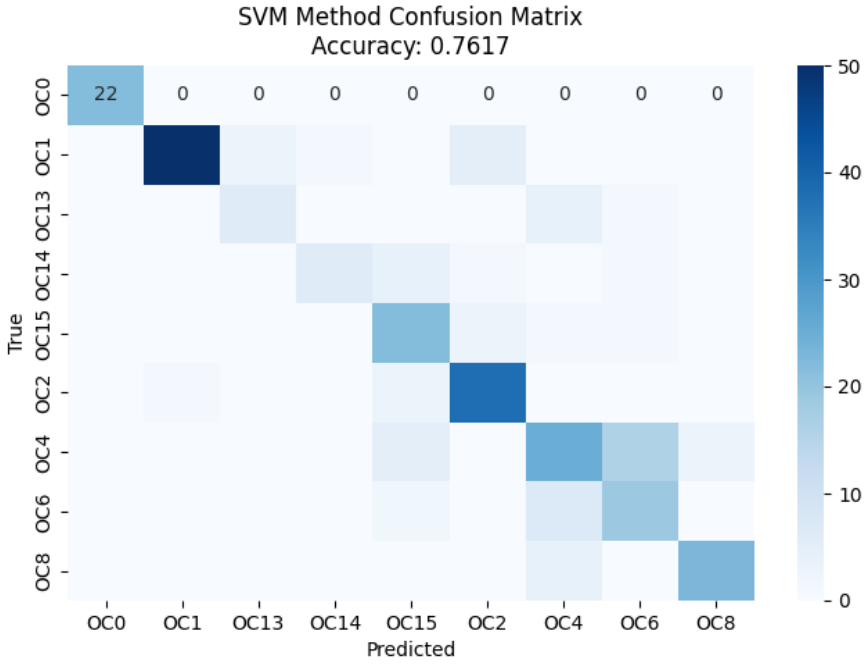


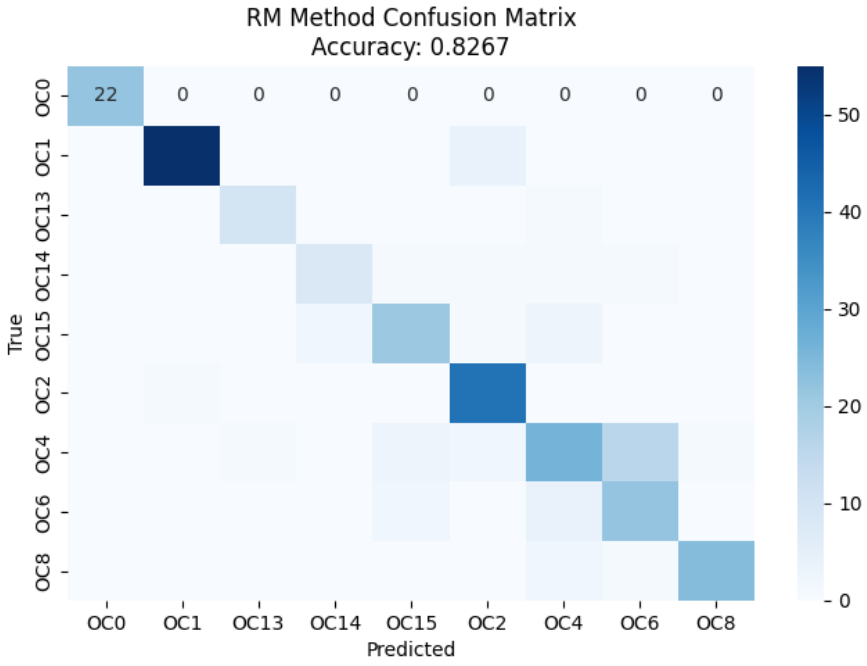
工况 0、1、2、4、8、13、14、15





工况 0、1、2、4、6、8、13、14、15





全 17（16 种故障、1 种正常）种工况