第 5 章

异 常 检 测

设备预测的/预设的(predictive/prescriptive)AI 生命周期从数据采集设计开始算起[①]。数据要先经过诸如相关性和方差等因素的分析,才可以开始制造该设备。除了实验用的少量样机外,一般来说设备不会出现故障,也就不能作为机器学习模型了。大多数生产商依据设备的工作周期(duty cycle)来判断一台设备的状态好坏。设备的工作周期取决于设备运行时是否过热,或者是否触发了传感器的报警值。如果是,相应的数据就需要立即进行更深入的分析。对一个分析师来说,数据量之大可能令人望而生畏。分析师需要检视数以百万计的记录,这就像人们常说的大海捞针一样。分析师提早介入(analyst-in-the-middle)并使用异常检测技术可以有效地辅助发现设备的故障。异常检测是通过统计的、无监督或有监督的机器学习技术实现的。换句话说,通常情况下,分析师首先查看单个数据点,被检查的数据点通常为尖峰或低谷。然后,将多个数据点导入一个对数据进行聚类的无监督学习模型中,帮助数据科学家发现一组数值或模式何时出现了异常。最后,在掌握足够多的设备故障后,分析师就可以使用同样的机器学习方法应用于预测性维护了。一些机器学习算法,如孤立森林(isolated forest),更适用于异常检测,但其原理其实是一样的。

异常检测可以在为有监督学习采集到足够的数据之前进行,也可以成为持续监控解决方案的一部分。例如,异常检测可以应用于不同工厂的生产故障报警。当电气工程师将设备的物理设计交给工厂时,需要对**物料清单**(**Bill Of Material, BOM**)进行优化。简而言之,他们要将设计方案转换为生产组装更方便的或是更具成本效益的。大多数硬件设备的使用周期为十年。在此期间,设备的零部件可能已无法及时获得,这就意味着需要对 BOM 进行修改。同样,供货商的变更也会要求原设计的改变,因为要进行相应的 BOM 优化。异常检测可以帮助确定设备群中突发的新情况。

异常检测的方法很多。3.1 节的实用案例采用了流行的 K-means 异常检测算法确定某种食品的化学特征,这只是异常检测的一种方法。还有许多不同类型的异常检测方法。有些是专门针对某一特定机器,检测其在一段时间内的异常情况。有些异常检测算法使用有

① 译者注:原文句首有 diarization,疑为误录入。

监督学习来检测设备处于正常运行还是异常运行。很多设备还会受到当地环境或季节性的影响。本章要讨论的是如何将异常检测部署到边缘端。

本章将涵盖以下实用案例：

- 在 Raspberry Pi 和 Sense HAT 上使用 Z-Spikes；
- 使用自编码器检测标记数据中的异常；
- 对未标记数据集使用孤立森林算法；
- 使用 Luminol 检测时间序列异常；
- 检测受季节性影响的异常；
- 使用流分析法检测峰值；
- 检测边缘设备的异常。

5.1 在 Raspberry Pi 和 Sense HAT 上使用 Z-Spikes

单台设备的数据出现峰值(spike)或突然发生变化可能会触发警报。IoT 设备经常会受到运动或天气的影响，可能是一天中不同的时间段，亦或是一年中不同的季节。一个设备群或许散落在世界各地，试图对整个设备群获得清晰的了解是一个巨大的挑战。采用可以协同处理整个设备群的机器学习算法，使我们能够对每台设备都能单独进行处理。

Z-Spikes 的用例包括电池突然放电或温度突然升高等。人们常用 Z-Spikes 来判断物体是否被推撞或突然开始振动，Z-Spikes 还可用于查看泵是否出现了堵塞。由于 Z-Spikes 在非同源(non-homologous)环境中表现出色，因此通常是边缘部署的最佳选择。

5.1.1 预备工作

本实用案例将在带有 Sense HAT 的 Raspberry Pi 上部署 Z-Spikes。对于 IoT 的初学者来说，此案例中的硬件本身是一些相当常见的开发板和传感器。实际上，初学者甚至可以将代码发送到国际空间站(International Space Station)，以便在他们的 Raspberry Pi 和 Sense HAT 上运行。如果手头没有这些装备，在 GitHub 资源库中可以找到替代代码来进行仿真。

Raspberry Pi 接通电源并连接 Sense HAT 后，还需要安装 SciPy。一般来说，Python 中所有的安装都可以通过 pip 命令完成，但本例需要通过 Linux 操作系统进行安装，为此，请在终端窗口运行以下命令：

```
sudo apt update

apt - cache show python3 - scipy
sudo apt install  - y python3 - scipy
```

通过 pip 命令安装 NumPy、Kafka 和 sense_hat。还需要在计算机上搭建 Kafka，1.6 节的实用案例中有详细说明。因为在 Raspberry Pi 上搭建 Kafka 所需的内存太大，所以建议在计算机上进行搭建。

Raspberry Pi 还需要连接到显示器、键盘和鼠标。开发者的工具菜单提供了 Python 编辑器，读者还需要知道提供 Kafka 服务的 IP 地址。

5.1.2 操作步骤

本实用案例的操作步骤如下。

(1) 导入库：

```
from scipy import stats
import numpy as np from sense_hat
import SenseHat
import json
from kafka import KafkaProducer
import time
```

(2) 等待 Sense HAT 在操作系统注册：

```
time.sleep(60)
```

(3) 初始化变量：

```
device = "Pi1"
server = "[the address of the kafka server]:9092"
producer = KafkaProducer(bootstrap_servers = server)
sense = SenseHat()
sense.set_imu_config(False, True, True)
gyro = []
accel = []
```

(4) 创建 Z-score()辅助函数：

```
def zscore(data):
    return np.abs(stats.zscore(np.array(data)))[0]
```

(5) 创建一个辅助函数 sendAlert()：

```
def sendAlert(lastestGyro,latestAccel):
    alert = {'Gyro':lastestGyro, 'Accel':latestAccel}
    message = json.dumps(alert)
    producer.send(device + 'alerts',key = bytes("alert",
                                                encoding = 'utf - 8'),
                  value = bytes(message, encoding = 'utf - 8'))
```

(6) 创建辅助函数 combined_value()：

```
def combined_value(data):
    return float(data['x']) + float(data['y']) + float(data['z'])
```

(7) 运行函数 main()：

```
if __name__ == '__main__':
    x = 0
    while True:
        gyro.insert(0,sense.gyro_raw)
        accel.insert(0,sense.accel_raw)
        if x > 1000:
            gyro.pop()
            accel.pop()
        time.sleep(1)
        x = x + 1
        if x > 120:
           if zscore(gyro) > 4 or zscore(accel) > 4:
               sendAlert(gyro[0],accel[0])
```

5.1.3 工作机理

本算法是在检查最后一条记录的值是否超过了前面 1000 条的标准差(σ)的 4 倍。4σ 表示每 15787 条记录中有 1 个异常，或者每 4h 会发生一次异常。如果把它改为 4.5σ，那就意味着每 40h 会发生一次异常。

通过导入 SciPy 进行 Z-score 评估，导入 NumPy 进行数据处理。然后将脚本添加到 Raspberry Pi 的启动程序中，这样只要电源重启，脚本就会自动运行。机器还需要等待外设，例如 Sense HAT 的初始化等。60s 的延迟可以让操作系统在尝试初始化 Sense HAT 之前检测到它。然后对变量进行初始化，这些变量包括设备名称、Kafka 服务器的 IP 地址以及 Sense HAT。然后启用 Sense HAT 的内部测量单元(**Internal Measuring Unit，IMU**)，禁用罗盘(compass)并启用陀螺仪(gyroscope)和加速度计(accelerometer)。最后，创建两个数组存储数据。

创建一个辅助函数 Z-cores()，可以在其中输入一个数组作为 Z-cores 的返回值。还需要一个可以发送警报的函数。函数 sense.gyro_raw()可以获取最新的陀螺仪和加速度计读数，并存放在 Python 对象中，还可以将其转换为 JSON 格式。创建一个 UTF-8 字节编码的键值，对消息的有效载荷进行编码。

创建一个 Kafka topic 名称，将密钥和消息发送到该 topic。通过__main__查看是否正在命令行解释器(command shell)中运行当前文件。如果是，将计数器 x 设置为 0，开始进

行无限循环。随后开始导入陀螺仪和加速度计的数据。检查数组中是否含有 1000 个元素。如果是，删除数组中的最后一个值，以便保持数组的规模。然后，计数器累加，保存 2min 的数据。最后，检查是否超过了数组中 1000 个值的标准差的 4 倍，如果是，就发送警报。

下面的实用案例会创建一个消息发送器和接收器，这是查看设备的一个很好的方法。如果想对整个设备群进行异常检测，还需要创建一个 Kafka producer 消息，以便在循环的每一次迭代中发送数据。

5.2 使用自编码器检测标记数据中的异常

如果手头有标记数据(labeled data)，就可以拿来训练模型以检测数据是否异常。例如，通过读取电机的电流就可以发现何时由于滚珠轴承故障或其他硬件故障对电机施加了额外的阻力。在 IoT 中，异常现象可以是事先已知的现象，也可以是从来没有遇到的新情况。顾名思义，所谓自编码器(autoencoder)就是可以将接收的数据进行编码并输出。通过异常检测，可以检视一个模型是否能够确定数据是异常的。本实用案例将用到 Python 对象检测库 pyod。

5.2.1 预备工作

本实用案例将使用由 Sense HAT 的运动传感器采集的数据。本章最后一个实用案例将展示如何生成该数据集。该标记数据集放在了本书配套的代码资源包里。案例中还使用名为 pyod 或 **Python Outlier Detection** 的 Python 离群值检测框架，它封装了 TensorFlow 并执行各种机器学习算法，如自编码器和孤立森林等。

5.2.2 操作步骤

本实用案例的操作步骤如下。

(1) 导入库：

```
from pyod.models.auto_encoder import AutoEncoder
from pyod.utils.data import generate_data
from pyod.utils.data import evaluate_print
import numpy as np
import pickle
```

(2) 使用 NumPy 数组将文本文件加载到 notebooks：

```
X_train = np.loadtxt('X_train.txt', dtype=float)
y_train = np.loadtxt('y_train.txt', dtype=float)
```

```
X_test = np.loadtxt('X_test.txt', dtype = float)
y_test = np.loadtxt('y_test.txt', dtype = float)
```

(3) 使用自编码器算法,将模型与数据集绑定:

```
clf = AutoEncoder(epochs = 30)
clf.fit(X_train)
```

(4) 得到预测分数:

```
y_test_pred = clf.predict(X_test) # outlier labels (0 or 1)
y_test_scores = clf.decision_function(X_test) # outlier scores
evaluate_print('AutoEncoder', y_test, y_test_scores)
```

(5) 保存模型:

```
pickle.dump( clf, open( "autoencoder.p", "wb" ) )
```

5.2.3 工作机理

首先,导入 Python 对象检测库 pyod。然后导入 NumPy 用于数据处理。导入 Pickle 用于保存模型。使用 NumPy 加载数据。训练模型获得预测分数并保存模型。

自编码器由输入端获取数据,其较小的隐含层可以减少节点的数量,也迫使它降低了维度。自编码器的目标输出就是它的输入,这就能够用机器学习训练一个模型并判断是否发生了异常。通过计算某个值与训练好的模型到底相差多远就可以判断是否发生异常了。图 5-1 从概念上给出了数据是如何被编码成一组输入的,然后在隐含层中降低其维度,最后数据输出到一个更大的输出集合。

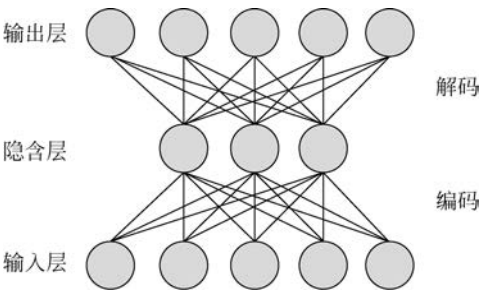

图 5-1 自编码过程

5.2.4 补充说明

模型训练完成后,还需要知道当处于何种级别时需要发送警报。在训练时,通过设置

contamination(代码如下)确定数据中触发警报功能所需的离群值比例:

```
AutoEncoder(epochs = 30, contamination = 0.2)
```

也可以用如下的代码改变 regularizer 的值。regularizer 用于平衡偏差和方差,以防止过拟合或欠拟合:

```
AutoEncoder(epochs = 30, l2_regularizer = 0.2)
```

还可以调整神经元的数量、损失函数或优化器。在数据科学中,这类操作通常被称为超参数(hyperparameter)的改变或调整。调整超参数可以影响成功指标(success metrics),从而改善模型。

5.3 对未标记数据集使用孤立森林算法

孤立森林是一种流行的用于异常检测的机器学习算法,可以帮助处理存在重叠值的复杂数据模型,是一种集成回归(ensemble regression)。它不像其他机器学习算法那样使用聚类或基于距离的算法,而是将离群的数据点与正常数据点分开。它通过建立一棵决策树,并根据数据所在的路径决策树中的节点数遍历来计算分数。换句话说,它通过计算所遍历的节点数量来确定结果。一个模型中训练过的数据越多,孤立森林需要遍历的节点就越多。

与前面的实用案例类似,使用 pyod 轻松地训练模型,此案例使用 GitHub 资源库中的 Sense HAT 数据集。

5.3.1 预备工作

如果已经实现了前面自编码器的实用案例,那么本案例就万事俱备了。本实用案例使用 pyod 作为对象检测库。训练数据集和测试数据集都可在本书 GitHub 资源库中找到。

5.3.2 操作步骤

本实用案例的操作步骤如下。

(1) 导入库:

```
from pyod.models.iforest import IForest
from pyod.utils.data import generate_data
from pyod.utils.data import evaluate_print
import numpy as np
import pickle
```

(2) 上传数据:

```
X_train = np.loadtxt('X_train.txt', dtype = float)
y_train = np.loadtxt('y_train.txt', dtype = float)
X_test = np.loadtxt('X_test.txt', dtype = float)
y_test = np.loadtxt('y_test.txt', dtype = float)
```

(3) 训练模型:

```
clf = IForest() clf.fit(X_train)
```

(4) 根据测试数据进行评估:

```
y_test_pred = clf.predict(X_test) # outlier labels (0 or 1)
y_test_scores = clf.decision_function(X_test)
print(y_test_pred)

# evaluate and print the results
print("\nOn Test Data:")
evaluate_print('IForest', y_test, y_test_scores)
```

(5) 保存模型:

```
pickle.dump( clf, open( "IForest.p", "wb" ) )
```

5.3.3 工作机理

首先,导入 pyod,然后导入 NumPy 用于数据处理,导入 Pickle 用于保存模型。接下来,进行孤立森林训练并评估结果。可以得到两种不同的结果:第一种是用 1 或 0 分别表示正常或异常;第二种是给出一个测试分数。最后,保存模型。

孤立森林算法使用基于树的方法对数据进行分割。数据的聚类程度越高,细分程度就越高。孤立森林算法通过计算到达密集分段区所需遍历的分段数量来寻找不属于密集分段区的数据。

5.3.4 补充说明

异常检测是众多分析技术中的一种,在该方法中,可视化技术可以确定需要使用哪个超参数或哪种算法。Scikit-learn 在其网站上给出了相关例子。本书 GitHub 资源库中也有这方面的参考资料。图 5-2 是一个在某玩具数据集上使用多种算法和设置进行异常检测的例子。在异常检测中没有哪种方法是最好的方法,只能说是对当前问题最有效的方法。

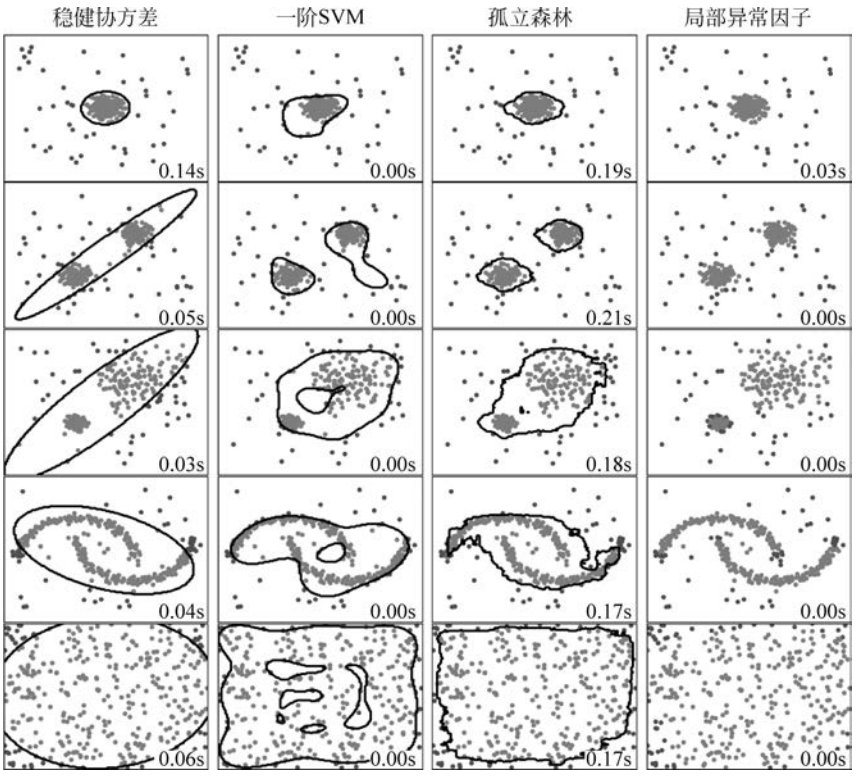

图 5-2　在某玩具数据集上使用多种算法和设置进行异常检测

5.4　使用 Luminol 检测时间序列异常

Luminol 是由 LinkedIn(领英)发布的一种时间序列异常检测算法。它使用位图检查数据集中有多少稳健的检测策略会发生漂移(drift)。该方法属于轻量级,但可以处理大规模的数据。

本案例使用一个来自美国芝加哥市的公开 IoT 数据集。芝加哥市采用 IoT 传感器检测湖泊的水质。在把数据集转换为异常检测所需的正确格式之前,还需要对其进行一些处理,因此将使用 prepdata.py 文件从某个湖泊中提取一个数据点。

5.4.1　预备工作

针对本实用案例,需要从本书 GitHub 资源库中下载 CSV 文件。然后需要安装 Luminol:

```
pip install luminol
```

5.4.2 操作步骤

本实用案例的操作步骤如下。

(1) 采用 prepdata.py 准备数据：

```
import pandas as pd

df = pd.read_csv('Beach_Water_Quality_ - _Automated_Sensors.csv',header = 0)

df = df[df['Beach Name'] == 'Rainbow Beach']
df = df[df['Water Temperature'] > -100]
df = df[df['Wave Period'] > -100]
df['Measurement Timestamp'] = pd.to_datetime(df['Measurement Timestamp'])

Turbidity = df[['Measurement Timestamp', 'Turbidity']]
Turbidity.to_csv('Turbidity.csv', index = False, header = False)
```

(2) 在 Luminol.py 中导入库：

```
from luminol.anomaly_detector import AnomalyDetector
import time
```

(3) 进行异常检测：

```
my_detector = AnomalyDetector('Turbidity.csv')
score = my_detector.get_all_scores()
```

(4) 打印异常情况：

```
for (timestamp, value) in score.iteritems():
    t_str = time.strftime('%y-%m-%d %H:%M:%S',
                          time.localtime(timestamp))
    if value > 0:
        print(f'{t_str}, {value}')
```

5.4.3 工作机理

在 dataprep Python 库中，只需要导入 Pandas，就可以得到 CSV 文件并将其转换为 pandas DataFrame。一旦有了 pandas DataFrame，通过 Rainbow Beach 进行过滤(本案例只用到 Rainbow Beach)。剔除异常数据，如水温低于－100℃的数据。把 time 字符串转换为 Pandas 可以读取的字符串，这样做可以确保在需要输出时，输出的是标准的时间序列格式。然后只选择需要分析的两列：Measurement Timestamp 和 Turbidity。最后，将文件保存为 CSV 格式。

创建一个 Luminol 文件，使用 pip 命令安装 luminol 和 time。然后在 CSV 文件上使用

异常检测器并返回所有的分数。如果分数项的值大于0，就返回分数。换句话说，只在有异常的情况下才会返回分数。

5.4.4 补充说明

除了异常检测，Luminol还可以进行相关性分析，这可以帮助分析师确定两个时间序列数据集是否相互关联。例如，来自芝加哥市的数据集反映了该市湖泊水质的各方面。可以将湖泊的数据进行对比，看看两个不同的湖泊在同一时间内是否具有相同的效应。

5.5 检测受季节性影响的异常

对于室外设备来说，其温度传感器白天的数据呈现上升趋势。同样，一台室外设备的内部温度在冬季一般会较低。不是所有的设备都会受到季节性的影响，但对那些受此影响的设备来说，选择一个能处理季节性影响的算法是很重要的。根据Twitter数据科学家发表的论文 *Automatic Anomaly Detection in the Cloud Via Statistical Learning*，**Seasonal ESD**就是一个不管季节怎么变化都能发现异常情况的机器学习算法。

对于本实用案例，将使用芝加哥市的湖水纯度数据集，所以需要导入5.4节实用案例中准备的数据文件。

5.5.1 预备工作

本实用案例中需要Seasonal ESD库。可以简单地通过以下pip命令进行安装：

```
pip install sesd
```

该数据集可以在本书的GitHub资源库中找到。

5.5.2 操作步骤

本实用案例的操作步骤如下。

(1) 导入库：

```
import pandas as pd
import sesd
import numpy as np
```

(2) 导入并处理数据：

```
df = pd.read_csv('Beach_Water_Quality_-_Automated_Sensors.csv',
                 header=0)
df = df[df['Beach Name'] == 'Rainbow Beach']
```

```
df = df[df['Water Temperature'] > -100]
df = df[df['Wave Period'] > -100]
waveheight = df[['Wave Height']].to_numpy()
```

(3) 进行异常检测：

```
outliers_indices = sesd.seasonal_esd(waveheight, hybrid = True,
                                     max_anomalies = 2)
```

(4) 输出结果：

```
for idx in outliers_indices:
    print("Anomaly index: {}, anomaly value: {}"\
          .format(idx, waveheight[idx]))
```

5.5.3 工作机理

本实用案例中，首先导入 NumPy 和 Pandas 用于数据处理，然后导入异常检测包 sesd。接下来为机器学习准备原始数据，剔除明显有问题的数据（例如那些不能正常工作的传感器），并将数据过滤成一列。最后，把该列数据输入 Seasonal ESD 算法。

与第一个实用案例中的 Z-score 算法类似，本实用案例使用的是在线方法。它使用 **Seasonal and Trend decomposition using Loess（STL）**分解方法作为异常检测前的预处理步骤。一个数据源可能会包含有某种趋势或季节性，如图 5-3 所示。

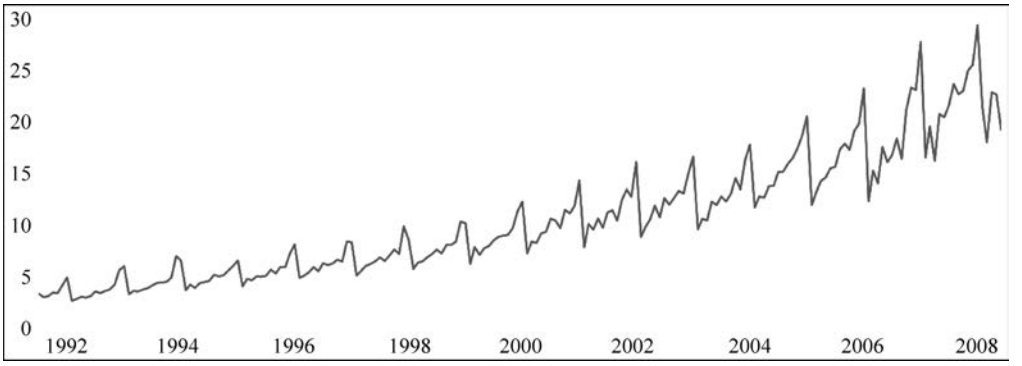

图 5-3 数据源包含的趋势性

通过分解方法可以独立地看待趋势和季节性（如图 5-4 的趋势图所示），这有助于确保数据不受季节性的影响。

Seasonal ESD 算法比 Z-score 算法更复杂。例如，Z-score 算法对安装在户外的设备就可能会发生误报（false positives）。

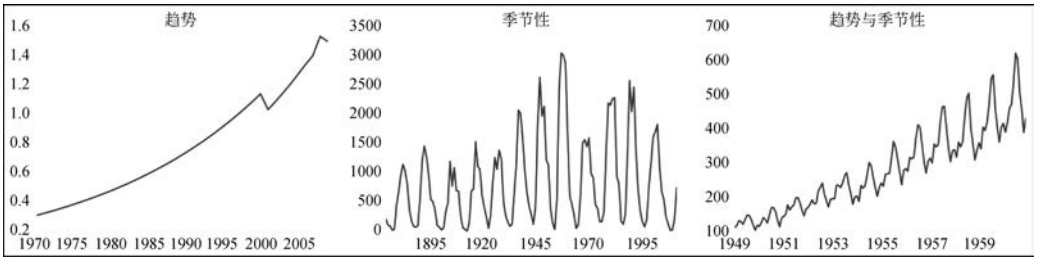

图 5-4　趋势和季节性图

5.6　使用流分析法检测峰值

流分析法(Stream Analytics)是一种使用 SQL 接口将 IoT Hub 与 Azure 内的其他资源连接起来的工具。流分析法将数据从 IoT Hub 转移到 Cosmos DB、storage Blobs、无服务器函数(serverless function)或其他一些可扩展选项。流分析法只有很少的内置函数,可以用 JavaScript 创建更多的自定义函数,异常检测就是其中之一。本案例使用 Raspberry Pi 将陀螺仪和加速度计数据流传到 IoT Hub,然后连接流分析,利用其 SQL 接口,仅输出异常结果。

5.6.1　预备工作

本实验需要 IoT Hub 并需要创建一个流分析作业。要实现这一点,进入 Azure 门户,通过 **Create new resource** 向导创建一个新的流分析作业,此时在 **Overview** 页面上有三个主要组成部分:输入、输出和查询。顾名思义,输入就是要输入的数据流,例如本案例中要输入的是 IoT Hub。要连接到 IoT Hub,单击 **Input** 按钮,选择 IoT Hub 的输入类型,再选择为本实用案例创建的 IoT Hub 实用案例。输出可能是一个类似 Cosmos DB 的数据库,也可能是一个功能应用程序 App,以便你可以通过任何数量的消息传递系统发送警报。为简单起见,本实用案例没有指定输出。出于测试目的,可以在 Stream Analytics 的 Query 编辑器上查看输出。

5.6.2　操作步骤

本实用案例的操作步骤如下。

(1) 导入库:

```
#device.py
import time
from azure.iot.device import IoTHubDeviceClient, Message
```

```
from sense_hat import SenseHat
import json
```

(2) 声明变量：

```
client = IoTHubDeviceClient.create_from_connection_string("your
device key here")
sense = SenseHat()
sense.set_imu_config(True, True, True)
```

(3) 获取连接设备的值：

```
def combined_value(data):
    return float(data['x']) + float(data['y']) + float(data['z'])
```

(4) 获取和发送数据：

```
while True:
    gyro = combined_value(sense.gyro_raw)
    accel = combined_value(sense.accel_raw)
    msg_txt_formatted = msg.format(gyro = gyro, accel = accel)
    message = Message(msg_txt_formatted)
    client.send_message(message)
    time.sleep(1)
```

(5) 创建一个 SQL 查询，使用 AnomalyDetection_SpikeAndDip 算法检测异常情况：

```
SELECT
    EVENTENQUEUEDUTCTIME AS time,
    CAST(gyro AS float) AS gyro,
    AnomalyDetection_SpikeAndDip(CAST(gyro AS float), 95, 120, 'spikesanddips')
        OVER(LIMIT DURATION(second, 120)) AS SpikeAndDipScores
INTO output
FROM tempin
```

5.6.3 工作机理

要在 Raspberry Pi 上导入需要的库，需要先登录 Raspberry Pi 并使用 pip 命令安装 azure-iot-device 和 SenseHat。接下来，在该设备上创建名为 device.py 的文件，并导入 time、Azure IoT Hub、SenseHat 以及 JSON 库。进入 IoT Hub，并通过门户网站创建一台设备，获取连接字符串，在 **Your device key here** 处输入该字符串。初始化 SenseHat，并将内部测量单元设置为 True，初始化传感器。创建一个组合 x、y 和 z 数据的辅助函数。从传感器获取数据，并将其发送到 IoT Hub。最后等待 1s，再次发送该数据。

进入已经设置好的流分析作业，单击 **Edit query**。从这里创建一个公共表表达式(common table expression)。公共表表达式可以让复杂的查询更加简单。然后在 120s 的

时间窗口内,使用内置的峰值和低谷异常检测算法。使用快速编辑器(quick editor)测试数据流中的实时数据,并查看异常检测器给出的异常或非异常的结果分数。

5.7 检测边缘设备的异常

本章最后一个实用案例将在 Raspberry Pi 上使用 SenseHat 采集数据,在本地计算机上训练这些数据,在设备端部署一个机器学习模型。为了避免记录数据出现冗余,需要运行本章前面给出的自编码器或孤立森林的实用案例中的任意一个。

在 IoT 中使用运动传感器可以确保航运集装箱安全地运上船。例如,若能证明某个航运集装箱是在某特定港口掉落的,将会有助于保险索赔。还可以通过检测跌落或工人的不当操作,保障工人的安全。运动传感器还被应用于发生故障时容易产生振动的设备,例如洗衣机、风力涡轮机和水泥搅拌机等。

在数据采集阶段,需要安全地模拟跌落或不当的操作,也可以将传感器装在不平衡的洗衣机上。本书配套代码资源包中的数据包括正常工作的数据以及发生抖动(dancing)的数据,在本案例中,称为 **anomalous**。

5.7.1 预备工作

针对本实用案例,需要一个带有 Sense HAT 的 Raspberry Pi,可以通过启用 SSH 或使用 USB 驱动器从 Raspberry Pi 获取数据。在 Raspberry Pi 上,需要使用 pip 命令安装 sense_hat 和 NumPy。

5.7.2 操作步骤

本实用案例的操作步骤如下。

(1) 导入库:

```
#Gather.py

import numpy as np
from sense_hat import SenseHat
import json
import time
```

(2) 初始化变量:

```
sense = SenseHat()
sense.set_imu_config(True, True, True)
readings = 1000
gyro,accel = sense.gyro_raw, sense.accel_raw
```

```
actions = ['normal', 'anomolous']
dat = np.array([gyro['x'], gyro['y'], gyro['z'], accel['x'],
                accel['y'], accel['z']])
x = 1
```

(3) 等待用户输入以便启动：

```
for user_input in actions:
  activity = input('Hit enter to record ' + user_input + \
                   'activity')
```

(4) 采集数据：

```
x = 1
while x < readings:
    x = x + 1
    time.sleep(0.1)
    gyro,accel = sense.gyro_raw, sense.accel_raw
    dat = np.vstack([dat, [[gyro['x'], gyro['y'], gyro['z'],
                     accel['x'], accel['y'], accel['z']]]])
    print(readings - x)
```

(5) 将文件输出至磁盘并进行训练：

```
X_test = np.concatenate((np.full(800,0), np.full(800,1)), axis = 0)
y_test = np.concatenate((np.full(200,0), np.full(200,1)), axis = 0)
X_train = np.concatenate((dat[0:800,:],dat[1000:1800]))
y_train = np.concatenate((dat[800:1000],dat[1800:2000]))

np.savetxt('y_train.txt', y_train,delimiter = ' ', fmt = '' % 10.8f")
np.savetxt('y_test.txt',y_test, delimiter = ' ',fmt = '' % 10.8f")
np.savetxt('X_train.txt', X_train,delimiter = ' ', fmt = '' % 10.8f")
np.savetxt('X_test.txt',X_test, delimiter = ' ',fmt = '' % 10.8f")
```

(6) 使用移动硬盘将文件从 Raspberry Pi 复制到本地计算机。

(7) 使用孤立森林的实用案例训练一个孤立森林，并输出 pickle 文件。

(8) 将 iforrest.p 文件复制到 Raspberry Pi 上，并创建一个名为 AnomalyDetection.py 的文件，并导入库：

```
#AnomalyDetection.py
import numpy as np
from sense_hat import SenseHat
from pyod.models.iforest import IForest
from pyod.utils.data import generate_data
from pyod.utils.data import evaluate_print
import pickle
sense = SenseHat()
```

(9) 加载机器学习文件：

```
clf = pickle.load( open( "IForrest.p", "rb" ) )
```

(10) 创建 LED 的输出：

```
def transform(arr):
    ret = []
    for z in arr:
        for a in z:
            ret.append(a)
    return ret

O = (10, 10, 10) # Black
X = (255, 0 ,0) # red

alert = transform([
        [X, X, O, O, O, O, X, X],
        [X, X, X, O, O, X, X, X],
        [O, X, X, X, X, X, X, O],
        [O, O, X, X, X, X, O, O],
        [O, O, X, X, X, X, O, O],
        [O, X, X, X, X, X, X, O],
        [X, X, X, O, O, X, X, X],
        [X, X, O, O, O, O, X, X]
])

clear = transform([
        [O, O, O, O, O, O, O, O],
        [O, O, O, O, O, O, O, O],
        [O, O, O, O, O, O, O, O],
        [O, O, O, O, O, O, O, O],
        [O, O, O, O, O, O, O, O],
        [O, O, O, O, O, O, O, O],
        [O, O, O, O, O, O, O, O],
        [O, O, O, O, O, O, O, O]
])
```

(11) 预测异常情况：

```
while True:
    dat = np.array([gyro['x'], gyro['y'], gyro['z'], accel['x'],
                    accel['y'], accel['z']])
    pred = clf.predict(dat)
    if pred[0] == 1:
        sense.set_pixels(alert)
    else:
```

```
            sense.set_pixels(clear)
        time.sleep(0.1)
```

5.7.3 工作机理

案例中创建了两个文件：一个用于采集信息(命名为 Gather.py)；另一个用于检测设备的异常情况(命名为 AnomalyDetection.py)。

在 Gather.py 文件中，导入库并初始化 SenseHat，为将采集的读数的数量设置一个变量，获取陀螺仪和加速度计的读数，创建一个常规的匿名字符串数组，并设置陀螺仪和传感器的初始等级值。按照用户的操作循环执行，当用户想要记录正常运行数据(normal greetings)时，提示用户按下 Enter 键；当用户想要记录异常读数时，也提示他们按下 Enter 键。至此就可以采集数据了，同时还可以给用户反馈，让他们知道还需要采集多少个数据点。

以正常使用的方式运行该设备，例如在进行跌落检测时，就要先将其抱紧，然后，在下一个读取异常数据的循环中，抛落该设备。最后，创建用于机器学习模型的训练集和测试集。将数据文件复制到本地计算机中，按照 5.3 节的方式进行分析。最后得到用于 AnomalyDetection.py 文件的 pickle 文件。

创建在 Raspberry Pi 上使用的 AnomalyDetection.py 文件。然后加载 pickle 文件，也就是机器学习模型。从这里开始，要创建 alert 和 clear(not-alert)变量，以便在感应装置上切换 LED 显示。最后，循环运行，如果预测到设备行为异常，就在感应器上显示 alert 信号；否则，就显示 clear 信号。