大连理工大学物联网课程设计

标题占用一行

专　　业：_ 学生姓名：___ 学　　号：_

指导教师：___ 完成日期：成　　绩：_____

大连理工大学

Dalian University of Technology

目　　录

1 选题背景与项目成员介绍

现有大规模制冷设备中螺杆压缩机和制冷系统设备故障检测技术存在实时性 差、准确率不高等问题，严重制约工业制冷设备故障的实时检测和超前预知， 使得实时安全管控极为困难，成为当前行业的“卡脖子”问题，研制具有自 主知识产权的边缘计算智能装备对解决国外技术壁垒和技术封锁迫在眉睫[1]。针对此问题，本文基于大连市揭榜挂帅项目“基于 5G 的分布式大规模制冷设备实时管控的边缘智能装备研发与应用”进行物联网专业的课程设计与实践。本文集中于学习研究故障检测算法、参与项目设计流程、了解项目开发过程。

该项目旨在通过边缘计算与云计算的结合，实现对制冷装备状态的实时监测与预警。制冷设备产生的数据量庞大，故障种类多，为提高故障检测效率，减少资源浪费，降低云端的压力，采用在边缘侧对制冷设备海量运行流数据进行初步排查，将筛查出的潜在故障数据传输至云端进行二次分析的方式，实现故障数据的收集与分类，以便后续制定合适的检修策略。

1.1 项目目标

（1） 处理公开数据，并寻找合适的数据库进行数据存储与使用。
（2） 基于公开数据集，学习并构建边缘侧二分类算法模型。
（3） 基于公开数据集，学习并构建云端多分类算法模型。
（4） 基于数据分类方法，学习并制作合适的展示界面。

1.2 项目分组

1.3 项目物理限制

由于疫情原因及学校时间安排，本项目无法使用真实传感器，因此采用指导教师提供的公开数据集进行项目设计与实践。

此外，受到疫情影响，本次项目中项目组同学无法线下见面交流，同时，由于无法模拟真实的大规模数据传输的场景，因此在线上开展会议的同时，采用分组工作的方式进行设计的实现。

2 项目需求及设计思路

2.1 展示页设计思路

一个好的界面设计应遵守以下规则：

1. 界面设计的原则是界面直观，对用户透明：用户接触软件后对界面上对应的功能一目了然。

2. 界面设计保持一致性。

3. 界面布局合理。

4. 输入预设的温差值，随后将数据发送至后台，经过计算后，将结果反馈到文本框中。

基于上述规则，在师兄提供的原型模板上进行功能增加，从而达到能够动态展示的程度。图 2.1 为师兄提供的原型。

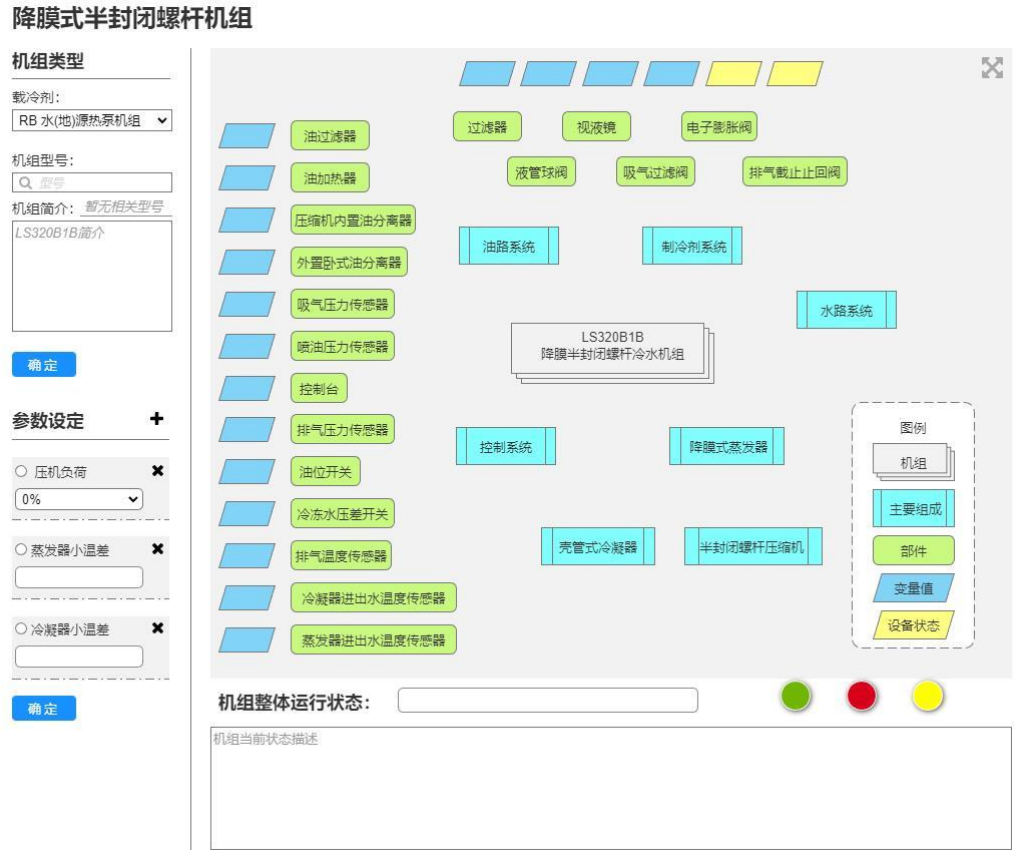

图 2.1 prototype

2.2 数据库的需求分析

制冷设备的运行数据具有数据量大的特点，且健康监测系统对于数据的高速访问要求较高，因此传统的关系型数据库并不适用来存储运行数据。所以本项目对于数据库的

需求是可以处理大量数据的高访问负载。根据项目要求，项目成员简单了解了国内的分布式数据库产品，对比了 GaussDB，TDSQL 和 TIDB 三款国产分布式数据库，最后选定了近几年较为常用的 TiDB。TiDB 是 PingCAP 公司自主设计、研发的开源分布式关系型数据库，是一款同时支持在线事物处理和在线分析处理的融合型分布式数据库。

2.2.1 整体架构

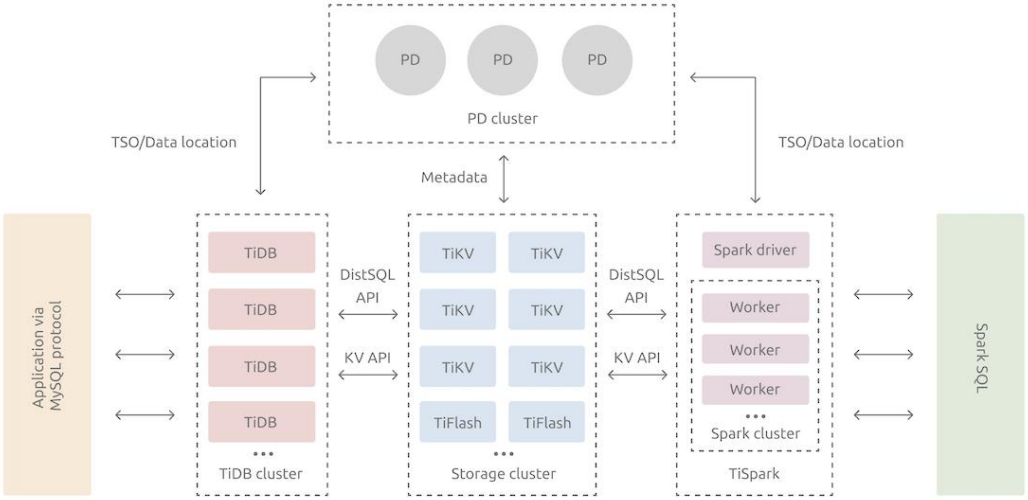

图 2.2 TiDB 架构图

1. TiDB Server

TiDB Server 是 TiDB 架构中的 SQL 层，即负责接收客户端连接的部分。该层本来是无状态的，在实践中可以同时启动多个 TiDB 实例，通过负载均衡组件对外提供统一的接入地址，客户端的连接可以均匀的分摊在多个 TiDB 实例上以达到负载均衡的效果。TiDB Server 本身并不存储数据，只是解析 SQL，将实际的数据读取请求转发给底层存储节点。图 2.2 为 TiDB 的架构图。

2. PD(Placement Driver) Server

PD 是整个 TiDB 集群的元信息管理模块，负责存储每个 TiKV 节点实时的数据分布情况和集群的整体拓扑结构，并为分布式事务分配事务 ID。PD 不仅存储元信息，同时还会根据 TiKV 节点实时上报的数据分布状态，下发数据调度命令给具体的 TiKV 节点。PD 本身由知道 3 个节点组成，拥有高可用的能力。

3. 存储节点

TiDB 的存储节点有两种类型，分别是 TiKV Server 和 TiFlash。

TiKV Server 是一个分布式的提供食物 Key-Value 存储引擎。存储数据的基本单位是 Region，每个 Region 负责存储一个 Key Range 的数据，每个 TiKV 节点负责多个 Region。

TiFlash 是一类特殊的存储节点。和普通 TiKV 节点不一样的是，在 TiFlash 内部，数据是以列式的形式进行存储，主要的功能是为分析型的场景加速。

2.2.2 存储与计算

TiKV 节点主要以键值对的方式保存数据，并且提供有序遍历方法，也就是说 TiKV 节点中的键值对是按照 Key 的二进制有序的。与其他持久化的存储引擎不同，TiKV 没有直接向磁盘上写数据，而是把数据保存在 RocksDB 中，由 RocksDB 负责具体的数据落地。这样的设计使得 TiKV 对单机引擎的各种要求都可以满足。最后，TiKV 使用 Raft 算法来保证在单机失效的情况下，数据不丢失也不出错。TiKV 利用 Raft 来做数据复制，每个数据会落地为一条 Raft 日志，通过 Raft 的日志复制功能，将数据安全可靠地同步到复制组的每一个节点中。

TiDB 在 TiKV 提供的分布式存储能力基础上，构建了艰巨优异的交易能力与良好的数据分析能力的计算引擎。对于分布式数据库来说，在扫描数据的时候，每一行都要想从存储节点中读取出来，那么如果数据量很大，开销也会很大，而且对于不符合查询要求的行，其实可以不进行读取。为了解决这些问题，计算应该尽量靠近存储节点。首先，SQL 语句中的为此条件应被推到存储节点进行计算，这要只需要返回有效的行即可。然后，聚合函数也可以被推到存储节点，进行预聚合，每个节点只需返回函数结果即可，再由 SQL 层对存储节点的返回值进行计算。

2.3 分类检测算法的需求及相应的调整

由于本项目数据的结果是不同的故障类型，因此，采用数字类型去判断故障类型显然是不合适的，因为数字类型包含了大小关系。我们无法假定第三种故障要比第二十七种故障高级，因此，需要将故障类型调整为独立的数据结构，并利用这个数据结构去判断出错误的概率。本项目采用了独热编码的方式，将 33 种故障类型独立编为 33 种 one hot encoding。通过将不同位置置为 1，表示不同情况的故障。相较于 haffman 和简单二进制编码方式，独热编码的好处在于直观、简明，不需要进行额外运算。图 2.3 为独热编码的情况结果。

	Category Label_23.0	Category Label_24.0	Category Label_25.0	Category Label_26.0	Category Label_27.0	Category Label_28.0	Category Label_29.0	Category Label_30.0	Ca Lab
..	0	0	0	0	0	0	0	0	
..	0	0	0	0	0	0	0	0	
..	0	0	0	0	0	0	0	0	
..	0	0	0	0	0	0	0	1	

图 2.3 独热编码

2.4 随机森林模型实现边缘侧数据二分类

在边缘测，即制冷装备运行数据产生处，采用随机森林模型对制冷设备产生海量数据进行二分类，排查出潜在故障数据，并将故障数据传至云端进行具体分析。

2.4.1 模型简介

随机森林是集成学习中的 bagging 方法，是有多棵决策树构成的集成算法，随机森林中的不同决策树之间没有关联。随机森林进行分类任务时，森林中的每一棵决策树会对输入样本进行分类和判断，分别得到自己的分类结果，随机森林会将决策树分类结果多的分类当作最终结果。

2.4.2 调参思想

在机器学习中，用于衡量模型在未知数据集上的准确率的指标成为泛化误差。泛化误差与模型结构的复杂程度息息相关，如图 2.4 所示，若模型太简单，拟合能力不够，模型会欠拟合，若模型太复杂，模型泛化能力不够，模型就会过拟合。只有当模型的复杂度恰到好处时，才能达到泛化误差的最小值。

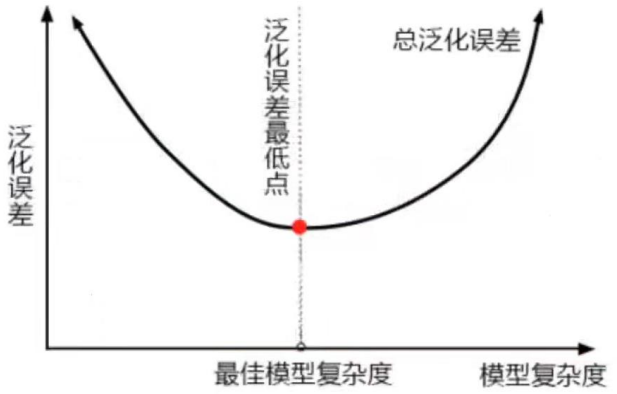

图 2.4 模型复杂度与泛化误差关系曲线

对于随机森林模型而言，树越茂盛，深度越深，枝叶越多，模型便越复杂，因此，随机森林是十分容易过拟合的模型，大部分的随机森林模型调参的目标都是减少模型的复杂度。在调整参数之前，需要判定模型处于图的哪一端，确定方向后才可进行模型参数的调整，直至找到泛化误差的最低点，达到模型的最优效果。

调参过程中，n_estimators 是对模型参数复杂度影响最大的参数，也是最先调整的参数，默认值为 100，表示森林中决策树的数量，通过调整使模型提升至平稳，不影响单个模型的复杂度。一般而言，先做出模型的学习曲线，根据曲线找到模型准确率最高的 n_estimators 值，再进行其他参数的调整。

max_depth 是对模型复杂度影响仅次于 n_estimators 的参数，表示树的最大深度。默认最高复杂度，调参时向复杂度简单的方向调整，使模型更简单。若调整后发现模型准确度降低，则模型已在图片左端，无需调整。

其次是对 min_samples_leaf 和 min_samples_split 的调整，该两项参数同样默认最高复杂度，调参时向简单的方向调整。若在调整 max_depth 时发现模型在图片左侧，则无需调整该两项参数。

次之是 max_features，是寻找最佳分割时考虑的特征数量，默认为最大特征数开平方。max_features 降低时，模型更简单，max_features 增大时，模型更复杂。max_features 是唯一既可以将模型调整简单，也可以将模型调整复杂的参数。

criterion 表示分裂阶段所用的标准，有 gini 和 entropy 可选，一般使用 gini。

3 项目过程

3.1 展示界面的工作流程、工作步骤及交互原理

3.1.1 工作流程

修改部分文本框：

#u15_input -- 压机负荷
#u10_input -- 蒸发器小温差
#u5_input -- 冷凝器小温差
#u85_input -- 接收机组整体运行状态
#u84_input -- 接收机组当前运行状态

修改按钮：

u107 -- 左下方 确定按钮
u106 -- 左上方 确定按钮

3.1.2 工作步骤

1. 向左下角传值，然后点击确定提交数据。
2. 读取左下角的三个输入框(u15, u10, u5)的值，并对其中的两个温差值做判断，打印故障信息。
3. 通过<u>**ajax**</u>请求，向后台发送数据。
4. 成功传输数据后，从后台读取计算后的结果，放在 data 中。
5. 通过 data 向，在页面下方的文本框（u85,u84）中，展示数据。

3.1.3 js 交互

采用 jquery 语法格式进行编写。截图为图 3.1.

```
//TODO -- 待写入后端地址
$("#btn_02").click(function () {
  // alert("点击按钮")
  var pressLoad = $("#u15_input option:selected").val();//压机负荷
  var evaporatorTemperatureDifference = $("#u10_input").val();//蒸发器小温差
  var condenserTemperatureDifference = $("#u5_input").val();//冷凝器小温差
  // alert(pressLoad)
  // alert(evaporatorTemperatureDifference)
  // alert(condenserTemperatureDifference)
  // $("#u85_input").val(""+pressLoad+" "+evaporatorTemperatureDifference+" "+condenserTemperatureDifference);
  // $("#u84_input").val(""+pressLoad+" "+evaporatorTemperatureDifference+" "+condenserTemperatureDifference);
  $("#u84_input").val("");
  if(evaporatorTemperatureDifference<2 || evaporatorTemperatureDifference>11.7){
    $("#u84_input").val("故障一");
  }
  if(condenserTemperatureDifference>-2.75 || condenserTemperatureDifference<-11){
    $("#u84_input").val($("#u84_input").val()+"故障二");
  }
  $.ajax({
    url:"###",//要发送的后台地址
    type:"post",
    data:{
      "pressLoad": pressLoad,
      "evaporatorTemperatureDifference": evaporatorTemperatureDifference,
      "condenserTemperatureDifference": condenserTemperatureDifference
    },
    dataType: "JSON",
    success:function (data) {//data为返回结果
      //TODO -- 接收计算后的结果
      $("#u85_input").val(data[0].xxx);
      $("#u84_input").val(data[0].xxx);
    }
  })
```

图 3.1 代码

3.2 数据库相关工作的设计与实现

3.2.1 数据预处理

在项目初级阶段，我首先根据对数据进行了预处理。我们获取的数据包含了一系列基准数据和不等类型不同等级的故障数据。用于训练和测试边缘侧二分类模型的数据集同时包含了基准数据和故障数据。基准数据为类别 0，故障数据为类别 1，混合打乱后形成二分类数据集用于训练和测试多分类模型的数据集只包含故障数据，因为我们已经在边缘侧将正常数据筛掉了。因此不同等级的故障数据将被一一划分为不同类别，并打乱混合后形成数据集。

此外，我还将数据集中的空白行，即无效数据进行了删除；对需要计算的空缺数据进行了计算并补全。

3.2.2 集群部署

在本项目中，由于资源的限制，我选择在单机上模拟部署生产环境集群。该部分的适用场景是在单台 Linux 服务器上体验 TiDB 最小的完整拓扑集群。首先，要准备一台安装了 CentOS 7.3 及以上版本,支持互联网访问的部署主机。因此,我将在安装了 CentOS 7.6 的虚拟机上进行集群部署。在开始部署之前,首先要关闭虚拟机的防火墙或开放 TiDB 集群所需要的端口。一下准备就绪后，部署集群的步骤。

1. 下载并安装 TiUP

   在终端中输入如下指令即可下载并安装 TiUP，该步骤要求虚拟机可以联网。

   ```
   curl --proto '=https' --tlsv1.2 -sSf https://tiup-mirrors.pingcap.com/install.sh | sh
   ```

2. 声明全局环境变量

   TiUP 安装完成后提示对应的 Shell profile 文件的绝对路径。将 ${your_shell_profile}修改为 Shell profile 的实际位置后执行指令即可。

   ```
   source ${your_shell_profile}
   ```

3. 安装 TiUP 的 cluster 组件

   指令如下：

   ```
   tiup cluster
   ```

4. 调整 sshd 服务的连接数限制

   该部分需要通过 root 用户进行。首先进入/etc/ssh/sshd_config 文件，将 MaxSessions 调整至 20，然后执行以下指令重启 sshd 服务。

   ```
   service sshd restart
   ```

5. 编写配置文件

   配置文件命名为 topo.yaml，主要包含了将要创建的集群的全局信息和节点信息。全局信息包括用户名，ssh 端口号，部署目录和数据目录等信息，节点信息包含了每个节点的 host 以及 port。最小的完整拓扑集群包含了 1 个 pd 节点，1 个 tidb 节点以及 3 个 tikv 节点。

6. 执行集群部署命令

   集群部署命令如下所示，参数<cluster-name>为集群名称，<tidb-version>表示设置的集群版本，可以通过 tiup list tidb 查看当前支持部署的 TiDB 版本。

tiup cluster deploy <cluster-name> <tidb-version> ./topo.yaml --user root -p

3.2.3 建库建表

在建库建表之前，首先要启动集群，利用 MySQL 访问 TiDB 数据库。由于 TiDB 与 MySQL 5.7 高度兼容，在 TiDB 中构建数据库和创建表的语句与 MySQL 语句类似。考虑到搜集的数据中并没有给出制冷设备的类型信息，本次设计数据库的过程中没有创建存储设备信息的表，只考虑了存储设备运行信息的表。具体的建库与建表语句如下所示。

```
-- 建库
CREATE DATABASE IF NOT EXISTS  Chiller_Database ;
-- 建表
CREATE TABLE opdata (
`TIME` datetime DEFAULT NOW(),
`FWC` decimal(10,5),
`TWE_set` int,
`TEI` decimal(10,5),
`TWEI` decimal(10,5),
`TEO` decimal(10,5),
`TWEO` decimal(10,5),
`TCO` decimal(10,5),
`TWCO` decimal(10,5),
`TCI` decimal(10,5),
`TWCI` decimal(10,5),
`TSI` decimal(10,5),
`TSO` decimal(10,5),
`TBI` decimal(10,5),
`TBO` decimal(10,5),
`Cond_Tons` decimal(10,5) generated always as (`FWC`*(`TCO`-`TCI`)/24) stored,
`Cooling_Tons` decimal(10,5) generated always as (`FWC`*(`TCO`-`TSI`)/24) stored,
`Shared_Cond_Tons` decimal(10,5) generated always as (`FWC`*(`TSO`-`TSI`)/24) stor
ed,
`Cond_Energy_Balance` decimal(10,5) generated always as (`Cond_Tons`+ `Cooling_Ton
s`+ `Shared_Cond_Tons`) stored,
`Evap_Tons` decimal(10,5) generated always as (`FWE`*(`TEI`-`TEO`)/24) stored,
`Shared_Evap_Tons` decimal(10,5) generated always as (`FWE`*(`TBI`-`TEO`)/24) stor
ed,
`Building_Tons` decimal(10,5) generated always as (`FWE`*(`TBO`-`TBI`)/24) stored,
`Evap_Energy_Balance` decimal(10,5) generated always as (`Evap_Tons`+ `Shared_Evap
_Tons`+`Building_Tons`) stored,
`kW` decimal(10,5),
`COP` decimal(10,5) generated always as ((`Evap_Tons`*12000)/(`kW`*3413)) stored,
`kW_Ton` decimal(10,5) generated always as (`kW`/`Evap_Tons`) stored,
`FWE` decimal(10,5),
`TEA` decimal(10,5),
`TCA` decimal(10,5),
`TRE` decimal(10,5),
`PRE` decimal(10,5),
```

```
`TRC` decimal(10,5),
`PRC` decimal(10,5),
`TRC_sub` decimal(10,5),
`T_suc` decimal(10,5),
`Tsh_suc` decimal(10,5),
`TR_dis` decimal(10,5),
`Tsh_dis` decimal(10,5),
`P_lift` decimal(10,5),
`Amps` int,
`RLA_per` int,
`Heat_Balance` decimal(10,5) generated always as (`Evap_Tons`*12000/3413+`kW`-`Con
d_Tons`*12000/3413) stored,
`Heat_Balance_per` decimal(10,5) generated always as (`Heat_Balance`*341300/(`Cond
_Tons`*12000)) stored,
`Tolerance_per` decimal(10,5) generated always as (10.5-0.07*`RLA_per`+(1500/(11.7
5*`RLA_per`))) stored,
`Unit_Status` int,
`Active_Fault` int,
`TO_sump` decimal(10,5),
`TO_feed` decimal(10,5),
`PO_feed` decimal(10,5),
`PO_net` decimal(10,5),
`TWCD` decimal(10,5),
`TWED` decimal(10,5),
`VSS` decimal(10,5),
`VSL` decimal(10,5),
`VH` int,
`VM` decimal(10,5),
`VC` int,
`VE` int,
`VW` int,
`TWI` decimal(10,5),
`TWO` decimal(10,5),
`THI` decimal(10,5),
`THO` decimal(10,5),
`FWW` decimal(10,5) generated always as (`FWC`*(`TCO`-`TSI`)/(`TWI`-`TWO`)) stored
,
`FWH` decimal(10,5) generated always as (`FWE`*(`TBO`-`TBI`)/(`THI`-`THO`)) stored
,
`FWB` decimal(10,5) generated always as (`FWC`-(`Shared_Evap_Tons`*24)/(`TSI`-`TSO
`)) stored
);
```

3.2.4 数据导入

TiDB 包含了将大数据一次性导入数据库的组件——TiDB Lightning，可以用来将 Dumpling 生成的文件，CSV 文件以及 Apache Parquet 文件导入到数据库中。图 3.2 所示为 TiDB Lightning 整体架构。

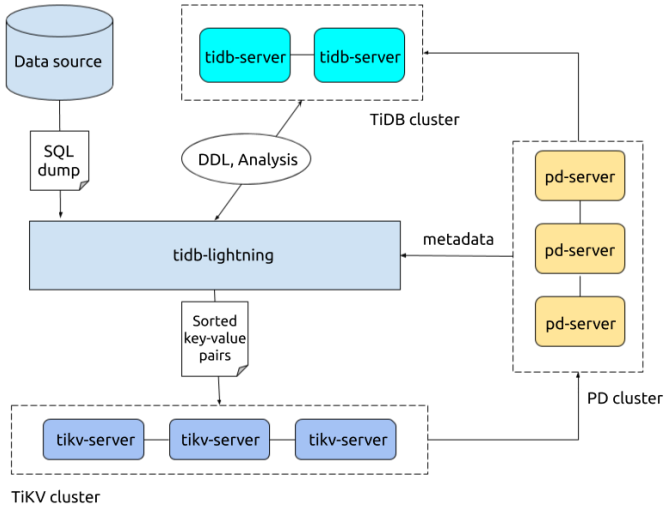

图 3.2 tidb-lightning 架构图

TiDB Lightning 目前支持两种导入方式，通过 backend 配置区分。不同的模式决定 TiDB Lightning 如何将数据导入到目标 TiDB 集群。

Physical Import Mode：TiDB Lightning 首先将数据编码成键值对并排序存储在本地临时目录，然后将这些键值对上传到各个 TiKV 节点，最后调用 TiKV Ingest 接口将数据插入到 TiKV 的 RocksDB 中。如果用于初始化导入，请优先考虑使用 Physical Import Mode，其拥有较高的导入速度。

Logical Import Mode：TiDB Lightning 先将数据编码成 SQL，然后直接运行这些 SQL 语句进行数据导入。如果需要导入的集群为生产环境线上集群，或需要导入的目标表中已包含有数据，则应使用 Logical Import Mode。

该迁移工具可以用来将大量数据导入 TiDB，由于本次项目中数据大小并不大，所以我选择了更为简便的 LOAD DATA INFILE 语句，具体导入语句如下所示：

```
LOAD DATA LOCAL INFILE '/home/swt/TIDB/data/data.csv ' INTO TABLE opdata FIELDS TE
RMINATED BY ',' ENCLOSED BY '' LINES TERMINATED BY '\r\n' IGNORE 1 LINES (FWC, TWE
_set, TEI, TWEI, TEO, TWEO, TCI, TWCI, TCO, TWCO, TSI, TSO, TBI, TBO, Cond_Tons, C
ooling_Tons, Shared_Cond_Tons, Cond_Energy_Balance, Evap_Tons, Shared_Evap_Tons, B
uilding_Tons, Evap_Energy_Balance, kW, COP, kW_Ton, FWE, TEA, TCA, TRE, PRE, TRC,
PRC, TRC_sub, T_suc, Tsh_suc, TR_dis, Tsh_dis, P_lift, Amps, RLA_per, Heat_Balance
, Heat_Balance_per, Tolerance_per, Unit_Status, Active_Fault, TO_sump, TO_feed, PO
_feed, PO_net, TWCD, TWED, VSS, VSL, VH, VM, VC, VE, VW, TWI, TWO, THI, THO, FWW,
FWH, FWB);
```

3.2.5 数据读写

在我们项目的总体设计中，模拟数据将被存入数据库中，预测前需要从数据库中读取。所以需要设计数据的读写函数。由于数据库部署在虚拟机上，所以需要实现从本地主机连接虚拟机上的数据库。连接之前，首先用 ping 命令和 telnet 命令检查本机是否可以与集群的 ip 地址和端口连接。

数据读取函数 getData()和数据写入函数 insert()代码类似，首先连接数据库，创建游标；接着执行相应的生 SQL 语句；最后关闭游标和连接。getData()函数比 insert 函数()多出了将数据写入 CSV 文件的步骤，因为模型的输入要求为 CSV 文件。

3.3 多分类检测算法的设计与学习过程

3.3.1 从张量开始

工欲善其事必先利其器，虽然现在版本的 python 支持了许多优秀的数据结构，但是在数据科学领域，还是需要采用一些封装好的工具包。本项目采用的是 pytorch，它的基本数据结构是张量（Tensor）。

张量是一个数组，它的本质是多维数组。它可以通过一个索引单独访问，也可以通过多个索引访问。比较 python 的 List 数据结构和 pytorch 的 Tensor 数据结构，可以发现他们的访问方式是很相似的，这为我们的工作带来了诸多便利。张量可以以多维度的方式在计算机中存储。以储存三角形三个顶点为例，我们既可以使用一维张量存储，即将坐标 x 轴存储在偶数索引内，将 y 轴存储在奇数内，也可以采用二位张量存储，如图 3.3.

```
points = torch.tensor([4.0, 1.0, 5.0, 3.0, 2.0, 1.0])
points
```

```
tensor([4., 1., 5., 3., 2., 1.])
```

```
float(points[0]), float(points[1])
```

```
(4.0, 1.0)
```

图 3.3 存储二位张量

张量的的索引方法有许多，采用 tensor_name[:]可以访问张量内的全部内容。在此基础上的使用方法可以扩展为 tensor_name[1:4]：访问第 1 到 3 个元素、tensor_name[1:]：访问第 1 到最后一个元素、tensor_name[:4]：访问开始元素到第 3 个元素、tensor_name[:-1]：访问开始元素到最后一个元素之前的元素、

tensor_name[1:4:2]：访问第 1 个到第 3 个元素，步长为 2.将使用方法扩展到高维，以二维为例，可以采用 tensor_name[1:, :]访问第 1 行后的所有行与所有列，如图 3.4.

```
some_list = list(range(6))
some_list[:]      # <1>
some_list[1:4]    # <2>
some_list[1:]     # <3>
some_list[:4]     # <4>
some_list[:-1]    # <5>
some_list[1:4:2] # <6>
```

图 3.4　张量索引

此外，我们也可以对张量进行命名和指定元素类型。张量的元素类型利用 dtype 属性在创建时指定，也可以后续利用其他方法转换。张量内的元素类型需要与模型相匹配才能够进行正常运算，目前常见类型有 float32、float64、float16、int8、uint8、int16、int32、int64、bool 等。

张量内部元素类型需要利用 dtype 属性进行指定。我们可以在创建张量的时候“torch.ones(10, 2, dtype = torch.double)”来指定，也可以在创建时调用张量的元素类型方法，如“torch.ones(10, 2).double()”、“torch.ones(10, 2).to(torch.double)”。

张量的元数据是大小、偏移量与步长。通过调用 torch 的 t()方法可以将张量矩阵进行转置。转置后的矩阵在计算机存储空间中并没有重新排列，因此如果调用转置后的张量进行运算，可能会出现张量非连续的错误，需要用 contiguous 方法将张量连续化。张量在计算机中的存储方法不同于 python 的 list 数据结构，张量是在连续的内存块中存储的，因此其计算非常快速。

3.3.2　机器学习

项目初期，本项目成员在 kaggle 平台上学习并运行了一些机器学习代码，了解了机器学习的机制：利用数学模型对输入值进行运算，并与输出值进行比对，通过与输出值的差异返回调整输入值的权重与偏移量，从而优化数学模型的准确率。

机器学习的流程要求先将数据处理成能够进行运算的格式，而后通过线性模型与激活函数的叠加实现将数据输出到有限的空间内，并利用反向传播机制将调整的参数传递回模型并进行调整，使用优化器进行梯度下降以最大程度地将调整参数的方向朝向最优的位置。

本项目数据储存在 xlsx 文件中，微软自带的 Excel 提供了将其另存为 csv 文件的功能。然而，这种 csv 文件缺失了一些数据集本应具有的功能，因此，需要在进行机器学习之前，对数据集进行读取。

通过使用数据处理工具 pandas，成功读入 csv 数据。然而，利用 pandas 读取的数据是具有独特的格式的，因此，需要利用 torch 的方法将 pandas dataframe 转换为 torch tensor。图 3.5 展示了将 dataframe 转换为 tensor 的过程。

```
import torch
X = train_loop_data.values
Y = train_loop_target.values
train_loop_data = torch.Tensor(X)
train_loop_target = torch.Tensor(Y)
```

图 3.5　dataframe 转换为 tensor

3.3.3　利用神经网络处理数据

神经网络的相关操作已经被封装在各种常见的机器学习库里面了。在学习这些代码的时候，本组选用了 pytorch 作为工具。Pytorch 中除了包含上述的张量操作外，还包含了一个专门用于神经网络的模块：torch.nn。

此模块内封装了许多专门用于神经网络的子模块，他们派生自 nn.Module，可以提供构建神经网络各种层的方法。神经网络的输入是数据集的特征，输出则为数据集应当判别的结果。在中间含有很多隐藏的层，这些隐藏层往往是一些线性函数与激活函数的嵌套。通过数学手段，已经有人证明，通过线性函数拟合一个问题是有很好的操作性的。Nn 模块中最简单的模型就是线性模型：nn.Linear(input, output).这个模型允许用户以 input 个数的输入参数输入，并经过线性运算，得到 output 个结果的输出。通过将线性函数与激活函数嵌套在一起，我们可以制作出自己的神经网络。Pytorch 中封装好的链接不同小模型的方法是 nn.Sequential，它允许用户将不同的方法结合到一起。如图 3.6 所示，neu 为制作的神经网络，其由线性模型和 Sigmoid 函数组成。通过将图 3.6 所示的神经网络作为模型输入给训练步骤，就可以对数据进行模型训练。

```
neu = nn.Sequential(
    nn.Linear(input_size, input_size*2),
    nn.Sigmoid(),
    torch.nn.Dropout(0.01),
    nn.Linear(input_size*2, 33),
    # nn.LogSoftmax(dim=1)
)
```

图 3.6　连接神经网络

对数据进行训练的过程应当是重复性的，因此需要一个循环来确保训练次数。循环的次数是从 1 次到 n 次，因此采用 for 循环的区间为 1 到训练次数+1.此外，训练的过程需要优化器来对权重、偏移量的调整进行指导，需要损失函数来计算模型的损失率，

判断模型是否向好的方向收敛。训练也同样需要传入训练数据、训练目标、验证数据和验证目标。图 3.7 所示是本项目使用的训练过程。通过 pytorch 封装好的 backward 方法，可以将优化器指导的方向反向传递给模型中，调整模型的权重和偏移量。

```
def training_loop(n_epoches, optimizer, model, loss_fn, train_set, val_set, train_ans, val_ans):
    for epoch in range(1, n_epoches+1):
        train_1 = model(train_set)
        print(train_1, train_ans)
        train_loss = loss_fn(train_1, train_ans)
        print("Train loss is ",train_loss)

        val_1 = model(val_set)
        val_loss = loss_fn(val_1, val_ans)

        optimizer.zero_grad()
        train_loss.backward()
        optimizer.step()

        if epoch==1 or epoch%1000 == 0:
            print(f"Epoch {epoch}, Training loss {train_loss},"f" Validation loss {val_loss}")
```

图 3.7　训练过程

3.4　BP 神经网络的学习与了解

在项目开发的前期，我们小组从零开始对深度学习特别是项目开发所用模型进行了深度学习，接下来我们将介绍有关于 BP 神经网络的基本信息，这将有助于对项目中心算法的理解。

BP 神经网络是一种多层的前馈网络，它与早期 NN 模型的区别在于，它加入了一种新的思想：负反馈调节，即它的误差会沿着与输入方向相反的方向进行传播，将输出信号与期望信号的误差进行分摊。

基本 BP 算法包括信号的前向传播和误差的反向传播两个过程。即计算误差输出时按从输入到输出的方向进行，而调整权值和阈值则从输出到输入的方向进行。正向传播时，输入信号通过隐含层作用于输出节点，经过非线性变换，产生输出信号，若实际输出与期望输出不相符，则转入误差的反向传播过程。误差反传是将输出误差通过隐含层向输入层逐层反传，并将误差分摊给各层所有单元，以从各层获得的误差信号作为调整各单元权值的依据。通过调整输入节点与隐层节点的联接强度和隐层节点与输出节点的联接强度以及阈值，使误差沿梯度方向下降，经过反复学习训练，确定与最小误差相对应的网络参数(权值和阈值)，训练即告停止。此时经过训练的神经网络即能对类似样本的输入信息，自行处理输出误差最小的经过非线形转换的信息。

BP 神经网络无论在网络理论还是在性能方面已比较成熟。其突出优点就是具有很强的非线性映射能力和柔性的网络结构。网络的中间层数、各层的神经元个数可根据具体情况任意设定，并且随着结构的差异其性能也有所不同。但是 BP 神经网络也存在以下的一些主要缺陷。

①学习速度慢，即使是一个简单的问题，一般也需要几百次甚至上千次的学习才能收敛。

②容易陷入局部极小值。
③网络层数、神经元个数的选择没有相应的理论指导。
④网络推广能力有限。

3.5 随机森林算法进行二分类的实现

加载数据，获取处理过的 csv 数据路径并读取数据。在这一步，可以使用 head()函数查看数据是否加载成功，使用 shape()函数查看数据量、特征数，使用 describe()函数查看标准差、最小值、四分之一分位数、二分之一分位数、四分之三分位数、最大值、均值等。如图 3.8 是经过处理的二分类部分数据分析表。

```
df='../input/dataset/binary.csv'
df=pd.read_csv(refrigeration_file_path)
```

	FWC	TWE_set	TEI	TWEI	TEO	TWEO	TCI	TWCI	TCO	TWCO
count	2.166300e+04	21663.000000	21663.000000	21663.000000	21663.000000	21663.000000	21663.000000	21663.000000	21663.000000	21663.000000
mean	2.604950e+02	44.456170	52.235420	52.347828	45.888112	45.806426	74.591438	74.453843	80.816188	80.604676
std	2.712802e+01	4.146663	5.296536	5.361058	4.868054	4.900027	7.561393	7.677094	8.246848	8.325557
min	3.868000e-45	39.000000	42.300000	28.000000	38.470000	28.000000	56.130000	-99.300000	56.380000	53.600000
25%	2.658000e+02	40.000000	48.400000	48.500000	41.010000	40.900000	67.270000	67.200000	73.700000	73.600000
50%	2.679000e+02	44.000000	51.920000	52.000000	45.350000	45.300000	75.300000	75.200000	80.900000	80.700000
75%	2.694000e+02	49.000000	56.010000	56.300000	50.020000	50.000000	80.910000	80.800000	87.990000	87.800000
max	2.754000e+02	50.000000	75.300000	79.000000	75.320000	75.400000	88.970000	89.000000	99.860000	104.700000

8 rows × 66 columns

图 3.8 二分类数据分析表（部分）

加载数据后，可进行简单的数据数据分析，进行可视化，得到各类型之间的相关系数表格，绘制数据的热力图、折线图等，对于数据进行初步认识。

对读取的数据进行特征列和标签列的划分，一般使用 X 标志特征列，使用 y 表示标签列。再将数据划分训练集和测试集，通常，测试集占总数据的 20%~30%，在此我将测试集划分为 20%。

```
X=df.drop('Category Label',axis=1)
y=df['Category Label']
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=10)
```

之后进行随机森林的建模，首先实例化随机森林分类器，采用 5 折交叉验证求得平均值，设置 random_state 为 10，得到默认参数时的分数为 0.9429。

```
model=RandomForestClassifier(n_estimators=100,random_state=10)
model.fit(X_train,y_train)
score=cross_val_score(model,X,y,cv=5).mean()
score
```

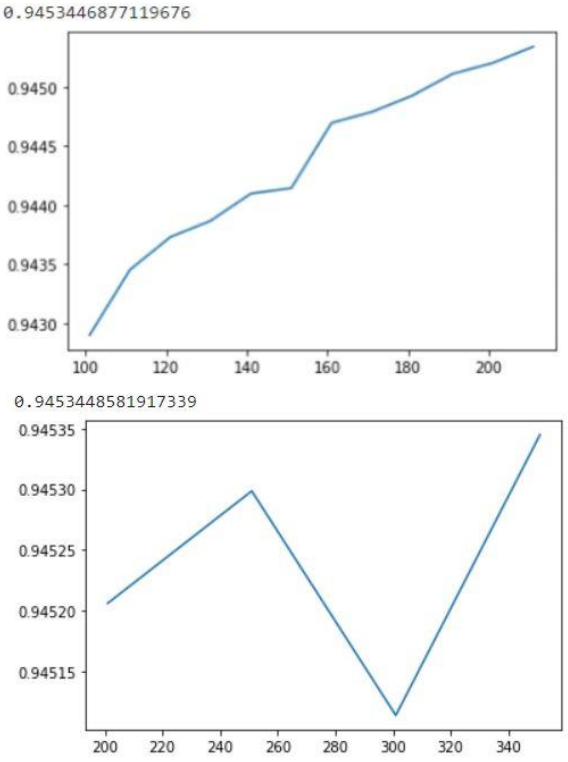

图 3.9 随机森林 n_estimators 学习曲线

随机森林的初步建模到此结束，接下来便是便是对参数进行调整，提高模型准确率。通常能够的提高准确率幅度很小。

绘制 n_estimators 的学习曲线，从曲线趋近平稳的部分找到使得准确率最高的 n_estimators 值。由于技术和时间等原因，目前暂未获得理想的 n_estimators 值，因此由图 3.9 暂且选定 210 为 n_estimators 值。

```
scores=[]
for i in range(100,220,10):
    model=RandomForestClassifier(n_estimators=i+1,random_state=10)
    score=cross_val_score(model,X,y,cv=5).mean()
    scores.append(score)
print(max(scores))
plt.plot(range(101,221,10),scores)
plt.show()
```

确定 n_estimators 之后使用网格搜索方式确定 max_depth。由于数据量较大，直接在 20~40 之间进行搜索。经过网格搜索后得到最佳的 max_depth 为 31，准确率为 0.9394，准确率较调整之前下降，说明模型在图的左侧，可能是欠拟合状态，因此采用默认的 max_depth，同时不再调整 min_samples_leaf 和 min_samples_split。

```
param_grid={'max_depth':np.arange(20,40,1)}
```

```
model=RandomForestClassifier(n_estimators=210,random_state=10)
GS=GridSearchCV(model,param_grid,cv=5)
GS.fit(X_train,y_train)
GS.best_params_
GS.best_score_
```

调整参数 max_features，可以提高模型的复杂度。max_features 默认参数为总特征值开平方，该项目使用的二分类数据共有 65 个特征，因此从 8~65 个选取最佳特征数。

```
param_grid={'max_features':np.arange(8,65,1)}
model=RandomForestClassifier(n_estimators=210,random_state=10)
GS=GridSearchCV(model,param_grid,cv=5)
GS.fit(X_train,y_train)
GS.best_params_
GS.best_score_
```

经过调整，最终确定的参数为 n_estimators 为 210，最终准确率为 0.9453，相较于最初上升了 0.0023。

```
model=RandomForestClassifier(n_estimators=210,max_features=,random_state=10)
score=cross_val_score(model,X,y,cv=5).mean()
score
```

4 遇到的困难与结果

4.1 数据库相关工作遇到的困难与结果

在本次数据库部署的过程中，遇到了许多问题，比如虚拟机内存分配过小导致卡顿，端口被占用导致集群部署失败，数据类型不一致导致 generated always as 语法不能正常使用。除了这些小问题之外，还有部署时追求的最小集群导致数据无法完全导入。

4.1.1 节点扩容

由于数据库部署在虚拟机上，容量等要素都十分受限，所以在导入数据的过程中，出现了“tikv disk full”的报错。为了解决这一问题，我对存储数据的 tikv 节点进行了扩容。

要扩容节点，首先要编写配置文件。该配置文件与创建集群时的配置文件大同小异，主要包括了目录信息，节点的 host 和 port 信息。在扩容之前，一般需要对集群进行风险检测和风险修复，防止扩容后的节点对集群的拓扑结构造成影响。风险修复完毕后，就可以执行以下指令进行节点扩容了。

tiup cluster scale-out <cluster-name> scale-out.yaml [-p] [-i /home/root/.ssh/gcp_rsa]

指令执行完毕后，可以执行以下指令对集群的拓扑结构进行查看，以此来确定扩容是否成功。

tiup cluster display <cluster-name>

从指令的执行结果中可以看出扩容成功，集群的 tikv 节点数从 3 个变成了 4 个。

```
192.168.206.133:20160  tikv        192.168.206.133  20160/20180
     linux/x86_64  N/A     /tidb-data/tikv-20160       /tidb-deploy/tikv-20160
192.168.206.133:20161  tikv        192.168.206.133  20161/20181
     linux/x86_64  N/A     /tidb-data/tikv-20161       /tidb-deploy/tikv-20161
192.168.206.133:20162  tikv        192.168.206.133  20162/20182
     linux/x86_64  N/A     /tidb-data/tikv-20162       /tidb-deploy/tikv-20162
192.168.206.133:20163  tikv        192.168.206.133  20163/20183
     linux/x86_64  N/A     /tidb-data/tikv-20163       /tidb-deploy/tikv-20163
```

图 4.1 集群 tikv 节点

4.2 神经网络部分遇到的困难与结果

虽然神经网络算法可以运行，然而，由于网络设计不够合理，内部计算可能出现除以零导致爆 nan 的问题。或许由于学习率设置不够合理，在梯度下降与反向传播的过程中，可以将参数权重回调，但效果并不明显。

```
tensor([[ 0.6198,  0.2768, -0.3518,  ...,  0.0873,  0.0953,  0.1309],
        [ 0.5960,  0.2964, -0.2437,  ...,  0.0303,  0.0284,  0.1982],
        [ 0.4705,  0.5419, -0.2450,  ..., -0.2159,  0.0218,  0.1815],
        ...,
        [ 0.4563,  0.5145, -0.2470,  ..., -0.1181,  0.1216,  0.2216],
        [ 0.4914,  0.4276, -0.2541,  ..., -0.1752,  0.0385,  0.2249],
        [ 0.4819,  0.5129, -0.2605,  ..., -0.1207,  0.1747,  0.1925]],
       grad_fn=<AddmmBackward0>) tensor([[0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 1.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.]])
Train loss is  tensor(nan, grad_fn=<DivBackward1>)
Epoch 1, Training loss nan, Validation loss nan
tensor([[nan, nan, nan,  ..., nan, nan, nan],
        [nan, nan, nan,  ..., nan, nan, nan],
        [nan, nan, nan,  ..., nan, nan, nan],
        ...,
        [nan, nan, nan,  ..., nan, nan, nan],
        [nan, nan, nan,  ..., nan, nan, nan],
        [nan, nan, nan,  ..., nan, nan, nan]], grad_fn=<AddmmBackward0>) tensor([[0.,
0., 0.,  ..., 0., 0., 0.],
        [0., 0., 1.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        ...,
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.],
        [0., 0., 0.,  ..., 0., 0., 0.]])
Train loss is  tensor(nan, grad_fn=<DivBackward1>)
```

图 4.2 结果

4.3 二分类随机森林的模型评估与遇到的困难

4.3.1 模型评估方法

本文所采用的模型评估方法是混淆矩阵与 ROC 曲线，下面将依次介绍。

1. 混淆矩阵

混淆矩阵时机器学习中很好的模型预测结果情形分析表，可以直观地了解到模型表现情况，通常以矩阵的行表示真实值，矩阵的列表示预测值。图 2.4 表示该随机森林模型的混淆矩阵，深色的两部分为预测正确，浅色两部分为预测错误。分类器进一步优化的方向应是减小左下角的数据，降低实际为故障，但预测为正常的数据。

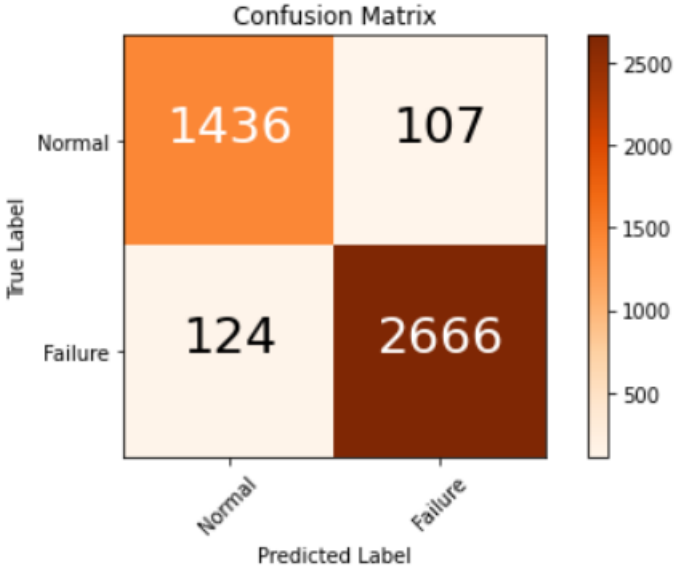

图 4.3 混淆矩阵

2. ROC 曲线

ROC 曲线，即受试者工作特征曲线，同样是模型评估的重要方式。如图 2.5，ROC 曲线的横坐标表示样本中为正常，但被模型分类为故障的数据，纵坐标表示故障数据被判定为故障的数据。ROC 曲线的线下面积 AUC 是量化反应 ROC 曲线衡量出的模型性能值，AUC 越大，表示分类器的性能越好。该随机森林模型的 AUC 为 0.9892。

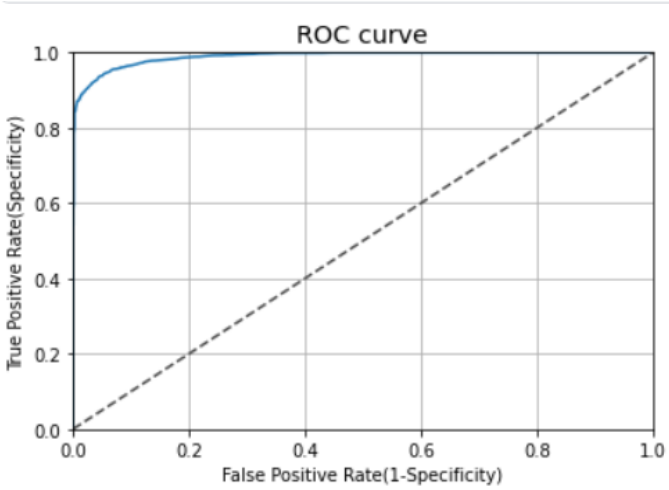

图 4.4 ROC 曲线

4.3.2 待优化的部分

由于多种原因限制，未能得到准确的 n_estimators 值，应需进一步缩小范围，获得更准确的参数。获得更合适的 n_estimators 之后，需要对其他参数进行再调整，并且重新进行评估。

虽然目前模型的准确率达到了 0.9453，但模型分类后的数据仍有很多不准确的部分，尤其是实际为故障数据，但模型分类为正确的数据，按照项目的设计，这一部分数据将不会传输至云端，无法进行故障判定，这将会导致可能存在的故障无法被检测到，这在实际应用中是十分严重的问题。

由于噪声的影响,随机森林分类器准确率无法达到 100%,模型处于泛化误差最低点，达到模型预测上限以后无法再进行优化，一定会存在错误分类的数据。若想提高检测的准确率，可以训练其他模型进行优化。

结　　论

参 考 文 献

[1] 刘选福. 基于 5G 的分布式大规模制冷设备实时管控的边缘智能备研发与应用[内部资料].[2021-04-25].