**分布式系统故障诊断系统设计文档**

|  |  |
| --- | --- |
| **团队姓名:** | 主函数 |
| **队长姓名:** | 汤嘉彧 |
| **队员姓名:** | 谢骏鑫 |
| **学 校:** | 南京航空航天大学 |
| **指导老师:** | 张道强 |

**二○二三年 6月**

# 引言

## 队伍介绍

我们是来自江苏省南京航空航天大学的主函数队，共由队长、一名队员和一名指导老师组成。

队长汤嘉彧，南京航空航天大学大二学生，主修计算机科学与技术专业，对深度学习计算机视觉方向兴趣较深，熟悉c++、python等编程语言。

队员谢骏鑫，南京航空航天大学大二学生，主修计算机科学与技术专业，有算法竞赛经历和前端开发经验，参与过操作系统开发相关竞赛。

指导老师张道强，南京航空航天大学教授，南京航空航天大学计算机科学与技术学院副院长。

## 开发背景

大数据时代，分布式系统成为信息存储和处理的主流系统。相对于传统系统而言，分布式系统更为庞大和复杂，故障发生的平均几率比较高，其运维的难度和复杂度大大提高。

因此，本赛题要求我们设计一个分布式系统的故障诊断系统，采用机器学习、深度学习等手段对分布式系统的故障数据进行分类，并高效地分析和识别故障类型，从而实现分布式系统故障运维的智能化，在快速恢复故障的同时大大降低分布式系统运维工作的难度，减少运维对人力资源的消耗。

## 设计要求

1. 设计一个能够对分布式系统的故障类型进行诊断分类的算法，并设计相应的故障分类模型。要求该模型能够对输入的真实业务数据进行故障分析与诊断，具有较高的准确率和较强的泛化性。
2. 分类模型的评价指标为f1-score，
3. 进行web平台的开发。该平台支持用户上传训练数据在线训练，训练结束后能够下载训练模型；该平台支持单条或批量测试样本上传和分类结果的下载，并能够将分类结果可视化；系统运行流畅；WEB平台的UI界面美观简洁、人机交互友好。

# 设计方案

## 2.1 开发环境与技术框架

我们采用前后端分离的系统开发技术框架，前端负责与用户交互和呈现功能，后端负责实现具体的故障诊断算法并为前端提供算法接口的支持。

前端WEB开发采用python…

后端人工智能算法部分采用了python开发环境，使用了pandas、numpy、sklearn等机器学习相关库以及基于pytorch的深度学习开发框架。

## 2.2 故障诊断算法介绍

故障诊断算法是本系统的核心部分，其主要内容可分为样本数据预处理、模型选择和搭建、模型训练和预测这三部分。

其中对样本数据进行预处理的先决条件是知晓样本数据的基本信息，即需要先通过数据挖掘手段对样本缺失值、样本中存在的特征相关、样本失衡程度等样本信息有一个大致的了解，从而才能对样本采取相对应的数据预处理方法。

模型的选择和搭建是故障诊断算法的核心模块，决定了故障分类的准确性和时效性。因此在模型的选择和搭建方面，我们需要同时考量模型的性能和大小，同时保证模型具有一定的泛化能力。

模型训练是将经过预处理后的训练集样本输入给已选取的模型，将模型处理后输出的结果与真实的样本标签值进行损失计算，利用梯度下降调整模型参数，实现训练模型的效果；模型预测则是将输入的验证集/测试集样本输入到已训练好的模型中，模型输出的值就是预测值，即诊断出的故障类型。最终可通过模型的验证集得分来评估模型的泛化能力。

以下对故障诊断算法中的几个部分进行依次介绍。

### 2.2.1 数据挖掘

数据挖掘是对训练集样本的数据特征进行探索性的观察和分析，从而对训练集样本的数据信息有所了解。在对10000个训练样本数据进行数据挖掘的过程中，我们主要进行了以下几个方面的数据信息探索：1、各个故障特征的数据类型。2、各个故障特征的均值、众数、分位数、标准差等基本数据特征。3、样本重复情况。4、样本失衡程度。5、样本缺失值情况。6、特征相关情况。

**1)各个故障特征的数据类型**

通过pandas库的内置函数df.info可观察到10000个样本中每一列的名称、缺失值情况以及数据类型。我们观察到，除了样本序号列和故障类型标签类的数据类型为int64，其余所有特征列的数据类型均为float64，即均为数值型数据，因此在接下来的数据预处理过程中无需对故障特征的数据类型进行转换。

**2)各个故障特征的基本数据特征**

通过pandas库的内置函数df.describe可观察到10000个样本中每一列数据的均值、标准差、最小值、25%分位数、50%分位数、75%分位数和最大值。其中通过每一个故障特征列的均值和标准差数值的比较，可观察到某些故障特征列的数值分布较不稳定(即样本标准差远大于样本均值)，可能部分样本中存在异常值，但考虑到这是来自于真实业务场景的数据，我们不会对样本数据轻易作出改动。另外可以寻找出标准差为0的故障特征列，即10000个样本的该故障特征取值都是相同的，因此可以认为这些特征列对最终的分类结果没有影响，在接下来的数据预处理过程中需要将这些特征列删除。

**3)样本重复情况**

将原始样本的sample\_id列删除后，可通过pandas库的内置函数df.duplicated()函数观察到样本的重复情况。经过筛查，10000个样本中存在着3704个重复样本。但由于这10000个样本来自于真实业务场景，这些重复样本本身可能就暗示着样本的分布情况，因此我们选择保留这部分重复样本。

**4)样本失衡情况**

样本不平衡是指在分类任务中不同类别的训练样本数目差别较大的情况。通过data\_clean['label'].value\_counts() / data\_clean.shape[0]语句可以查看训练样本的类别比例(data\_clean为10000个训练样本的DataFrame类型变量)，如图2.1所示。

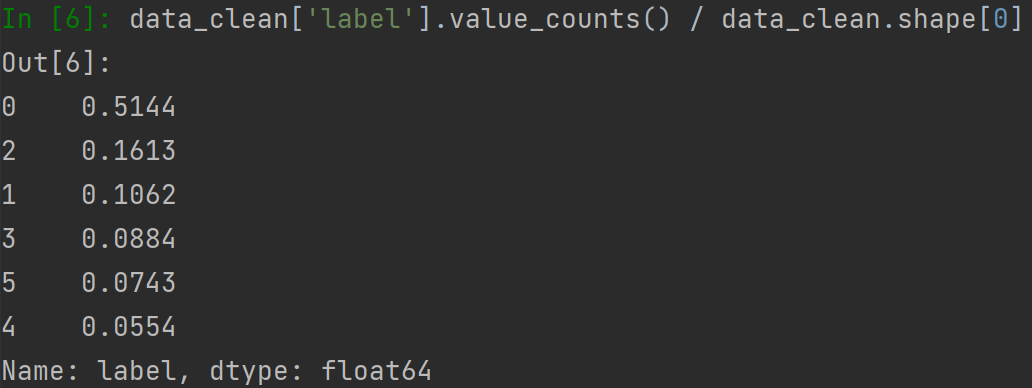


图2.1训练集样本的类别比例

根据图2.1中的类别比例，可见0类别的占比在50%以上，而4类别的占比仅仅为5.54%，存在一定程度的样本不平衡。因此在接下来的数据预处理过程中，可能需要采取相应的过采样或欠采样方法对样本进行处理。

**5) 样本缺失值情况**

在上述第1点中，我们通过df.info()函数可以初步观察到训练集样本的数据缺失情况。如果某一故障特征列的数据缺失比例在可接受范围内，则采取缺失值填充的方法进行解决；而如果某一故障特征列的数据缺失比例过高(如大于50%)，那么需要将这一特征列删除。

因此，我们需要对每一个故障特征列中缺失值的数目及所占比例有更为直观的了解，从而确定应该对存在数据缺失的特征列采取何种处理方式。我们通过以下函数来查看存在缺失值的特征列数，以及每一个缺失列中缺失值的数目和所占比例。

函数：

def missing\_values\_table(df): *# 输入：dataframe数据，输出：缺失总量即比例，降序输出  
 # 计算总的缺失值数量并降序处理* mis\_val = df.isnull().sum().sort\_values(ascending=False)  
 mis\_val = mis\_val[mis\_val > 0] *# 提取有缺失值的列  
 # 计算缺失值比例* percent = round(mis\_val \* 100 / len(df), 2)  
 mis\_info = pd.concat([mis\_val, percent], axis=1, keys=['Missing Values', 'Percent'])  
 *# 打印总结信息：总的列数，有数据缺失的列数* print("数据集共有 " + str(df.shape[1]) + " 列.\n" + "其中 " + str(mis\_info.shape[0]) +  
 " 列有缺失值")  
 *# 返回带有缺失值信息的dataframe* return mis\_info

训练集样本的缺失值情况如图2.2所示。

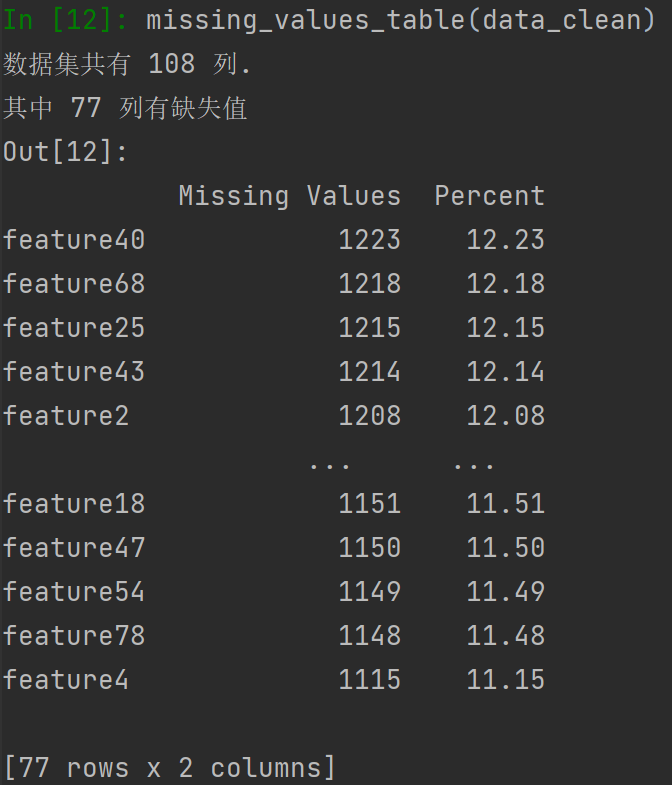


图2.2训练集样本的缺失值情况

根据图2.2，可以观察到缺失值占比最高的故障特征类是feature40，其缺失值占比为12.23%，在可接受范围内。因此对于这77个存在缺失值的特征列，我们都采取填充缺失值的方式进行处理。

**6) 特征相关情况**

当两个故障特征列具有很强的线性相关性时，两者对最终的分类结果的影响程度相近，只需保留其中与故障类别标签列线性相关性更强的特征列。通过pandas的内置函数df.corr()可以观察两两特征列之间的线性相关性，线性相关性的指标值在[-1,1]区间内，负数表示负相关，正数表示正相关，绝对值越大则相关性越强。我们通过以下函数来筛除原始训练集样本中的部分故障特征列，这些特征列满足：存在另一个特征列与该特征列的线性相关性指标大于某个特定阈值，且该特征列与故障类型标签列的线性相关性较弱。

函数：

def del\_corr\_feature(inx, iny, th):  
 *# 特征与目标的相关性* data = pd.concat([inx, iny], axis=1)  
 data\_corr = data.corr(numeric\_only=True).abs()[iny.columns[0]]  
 *# 特征与特征间的相关性* features\_corr = inx.corr(numeric\_only=True).abs()  
 size = inx.shape[1] *# 特征数* corr\_list = [] *# 列表的列表，用于存储[v, i, j]形式的列表，v表示特征i和j的相关系数且v>th* high\_corr\_fea = [] *# 存储相关系数超过th的特征名称  
 # 筛选相关系数超过th的特征* for i in range(0, size):  
 for j in range(i+1, size):  
 if abs(features\_corr.iloc[i, j]) >= th:  
 corr\_list.append([features\_corr.iloc[i, j], i, j])  
 *# 对corr\_list排序，其中关键字是相关系数的绝对值，排序方式为降序* sorted\_corr\_list = sorted(corr\_list, key=lambda xx: -abs(xx[0]))  
 cols = inx.columns  
 *# 遍历列表sorted\_corr\_list， 将其中待删除的特征加入到high\_corr\_fea列表中  
 # 将与目标特征相关系数较高的特征保留，较低的则待删除* for v, i, j in sorted\_corr\_list:  
 if data\_corr[cols[i]] >= data\_corr[cols[j]]:  
 high\_corr\_fea.append(cols[j])  
 else:  
 high\_corr\_fea.append(cols[i])  
 *# 列表去重* high\_corr\_fea = list(set(high\_corr\_fea))  
  
 *# 删除特征* inx = inx.drop(high\_corr\_fea, axis=1)  
 return inx

上述函数的参数中，inx为输入的带标签列的dataframe类型变量；iny为输入的标签列的dataframe类型变量；th是人为设定的阈值，只有当两个特征列的相关性高于th时，才认为这两个特征具有强相关性。函数的返回值为筛除了强相关性特征列之后的dataframe类型变量。

在查看特征相关情况之前，需要先对10000个样本进行缺失值的填充。由于这并不属于真正的数据预处理过程，因此缺失值填充方式的选择并不是太重要，这里我们选择KNN填充。在完成缺失值填充后，我们调用上述del\_corr\_feature函数，设定阈值th为0.5，即我们认为当线性相关性高于0.5时，两个特征列的相关性较强。经过处理后的样本与处理前的样本相比，特征列数不变，即不存在需要删除的强相关性特征列。因此在接下来的数据预处理过程中，我们也不需要考虑删除强相关性的故障特征列。

### 2.2.2 数据预处理

经过2.2.1中的数据挖掘步骤，我们对训练集样本的数据特征有了基本的了解，我们以此为参考，对训练集样本进行相应的数据预处理。

数据预处理的主要内容分为删除无关列、填充缺失值、处理样本不平衡、特征选择以及特征缩放。

**1)删除无关列**

训练集样本中的无关列为sample\_id列，该列仅仅表示样本序号，与样本的故障类型不存在任何联系，属于无关列，因此首先在训练集样本中删除sample\_id列。

**2)填充缺失值**

缺失值填充的方法主要有均值填充、众数填充、固定值填充、KNN填充、插值填充等。我们对上述缺失值填充方式均进行了尝试，最终选择了效果较好的KNN填充方式。

对于训练集样本中的每一个nan值，采用KNN填充方式就是对该nan值所在的样本，计算该样本到数据集中其他样本的欧式距离，选择其中距离最近的k个样本，用这k个样本对应特征的数值的平均值作为nan的填充值。其中如何衡量样本间的距离是KNN填充的一个重点，默认情况下是根据样本的所有特征去计算一个样本到另一个样本的欧式距离。但是对于本训练集而言，每一个样本都有107个故障特征列，如果根据这全部107个特征列去计算样本间的距离，时间开销会非常大。取而代之，我们观察发现feature22的特征列与故障类型标签列具有最强的线性相关性，因此我们以特征列feature22作为计算样本间距离的标准，即两个样本间的距离就是两个样本的特征feature22数值的欧式距离。

KNN填充算法中的邻域大小参数k作为超参数，需要人为调整与选择。经过调参，我们发现k=3时的填充效果最为理想，因此缺失值的填充方式为KNN填充，且参数k=3。

**3)处理样本不平衡**

在2.2.1的数据挖掘过程中，我们发现训练集存在一定程度的样本失衡问题。而如何对训练集进行平衡化处理，是一个需要反复调整和尝试的过程。

当我们没有对训练集进行任何的平衡化处理，即使用训练集中原始的10000个样本去对模型进行训练，之后采用验证集去检验模型效果，发现：0类样本的召回率很高，但精确率相对较低；而其余类样本的精确率较高，但召回率相对较低。结合训练集的类别比例，0类样本占比一半以上，这就导致训练出的模型在验证集上的分类结果会更偏向于0类，这进而导致1到5类样本容易被误判为0类，致使0类样本精确率低，而1到5类样本召回率低。

因此，我们采用重采样1到5类的方法，将训练集中1到5类的样本各复制一份以提高1到5类的类别比例。此时训练出的模型在验证集上的分类得分有所提升，样本失衡问题得到一定程度的改善。

**4)特征选择**

特征选择是对当前107个故障特征列中的部分特征列根据某些指标进行筛除。结合2.2.1中数据挖掘的结果，训练集样本中不存在相关性很强的特征列，但是存在一些方差为0的特征列。于是我们采用方差过滤法，将107个特征列中方差为0的3个特征列进行删除，这3个特征列分别为feature57、feature77和feature100。

我们还考虑将单值重复率高(即绝大部分的样本的该特征值都是相同的数值，只有少数样本的该特征值取值不同)的特征列进行筛除，但经过尝试后发现验证集上的分类得分并没有得到提升，因此最终选择保留这些特征列。

**5)特征缩放**

特征缩放，简而言之，就是将不同特征的数值量化到同一区间的方法。如果特征间的数值范围差异过大，则在采用欧式距离衡量两个特征间的距离时，会导致最终的距离仅仅取决于数值范围跨度最大的那个特征。而对数据集进行特征缩放处理之后，不仅能够消除量纲的影响，还能够显著提升模型的收敛速度。

特征缩放的一般方式有归一化和标准化。常见的归一化方式是MinMax归一化，我们假设某一特征列中特征点的最大值为maxi，最小值为mini，某一个特征点的值为x­，则经过归一化运算后该特征点的值为(x-mini)/(maxi-mini)，可见经过MinMax归一化处理后的特征点取值在[0,1]的区间范围之内。在标准化处理中，我们假设某一特征列中特征点的均值为meani，方差为vari，某一个特征点的值为x，则经过标准化运算后该特征点的值为(x-meani)/vari，可见经过标准化处理后，该特征列满足均值为0、方差为1的数据特征。我们对上述两种特征缩放的方法都进行了尝试，发现采用MinMax归一化会导致训练阶段模型难以收敛，而标准化则不会带来这一问题。我们认为这是因为，相较于归一化的将一列数据都缩放到[0,1]区间，标准化能够更好地保持样本间距和特征列的原有数据分布。因此我们对训练集采取标准化的特征缩放处理。

### 2.2.3 模型的选择与介绍

故障分类模型的选择是故障诊断系统的核心内容，模型的性能和大小影响着故障诊断算法的准确性和时效性。对于分类模型，现阶段常见的有机器学习框架中的逻辑回归(Logistic-Regression)、支持向量机(svm)、线性分类模型(LinearSVC)、K近邻(KNN)、朴素贝叶斯(Naive Bayes)、随机森林(random forest)、梯度提升分类树(Gradient Boosting Decision Tree)，以及深度学习框架中的一系列卷积神经网络模型。

在对上述这些模型进行选择时，我们需要找到一个用以评判模型分类性能的标准，从而在这些众多的模型中确定出最适合当前任务场景的模型。根据本赛题要求，算法模型的评测得分按照macro\_F1的标准进行计算，用以计算macro\_F1的函数如下：

def macro\_f1\_compute(y\_true, y\_pred):  
 macro\_P = precision\_score(y\_true, y\_pred, average='macro')  
 macro\_R = recall\_score(y\_true, y\_pred, average='macro')  
 macro\_F1 = 2 \* macro\_P \* macro\_R / (macro\_P + macro\_R)  
 return macro\_F1

在传入函数的参数中，y\_true为故障类型标签值，y\_pred为模型预测的故障类型值。函数的返回值就是当前预测结果的macro\_F1得分。

我们从两方面对一个模型的性能进行评估。一方面，我们将10000个样本的数据集划分为样本类别比例相近的训练集和测试集，并对训练集和测试集各自进行缺失值填充、特征选择和特征缩放。我们将数据集划分的步骤安排在了数据预处理步骤之前，是为了防止我们划分出的测试集中的信息泄露到训练集中，从而保证最终经过训练集训练得到的模型在测试集上的得分不会丧失参考价值。我们将经过数据处理后的训练集用于所选取模型的训练，训练完毕后，将测试集作为模型的输入，根据模型输出的预测故障类型值和测试集对应的故障类型标签值，计算得到macro\_F1得分，根据该得分高低评价模型的分类效果的好坏。但是仅仅依靠人为划分的训练集和测试集去评估一个模型的性能，并不能保证模型的泛化性和鲁棒性。因此另一方面，我们将10000个样本的数据集作为训练集，对该训练集进行2.2.2中的数据预处理。我们将经过处理的训练集用于所选取模型的训练，并计算所提供的1000个样本的验证集在训练好模型上的macro\_F1得分，根据该得分去评价当前模型的泛化性，并进一步认识当前模型的分类性能。

综合上述两方面的考量，我们最终选择一维卷积神经网络作为故障诊断所用的模型，主要原因在于其在人为划分的测试集和验证集上都有相对较高的macro\_F1得分，且表现稳定，具有较好的泛化能力。

对于逻辑回归、朴素贝叶斯、支持向量机、K近邻等传统机器学习模型，它们存在上限低、容易欠拟合的问题，各方面表现都不如树模型和神经网络模型。而对于随机森林、梯度上升分类树、XGBoost等树模型，它们在人为划分的测试集上表现优越，但在验证集上的表现却很糟糕。经过观察，我们发现验证集的样本类别比例几乎是6类均等，而10000个样本的训练集的样本类别比例则是0类占比一半以上。训练集和验证集类别比例的较大差异导致了树模型会将大量非0类故障识别为0类故障，

### 2.2.4 模型训练和预测

## 2.3 Web平台开发

本赛题的Web平台使用了非常经典的基于Python编写的一个Web框架——Django框架，在Django框架的支持下，我们根据大赛的要求完成了我们的Web平台的开发。大赛对于Web平台的要求除了运行流畅和UI界面简洁美观之外，还有两个大赛主题密切相关的要求，即能够支持用户上传训练数据并且在线训练，训练完成后可以下载对于的训练模型，这个要求我们将它视作训练模块，还有一个是能够单条或者批量上传测试样本，并将预测分类的结果在Web前端平台上可视化，并且支持下载分类结果，而这个要求则视作预测模块。因此整体上我们的Web需要实现的核心功能无法就是三点：上传数据、下载数据、信息可视化，接下来就分两点来分别阐述这两点的实现逻辑。

### 2.3.1 训练模块

训练模块需要实现的核心功能就是支持用户上传训练数据集并且能够在线进行训练，同时对于训练生成的模型提供下载功能，这一整个功能逻辑上就是一条完整的页面跳转链，在Django的设计框架中，我们对于用户上传的文件进行了模型化，并且同步到我们在开发网站中使用的数据库MySQL中，这样一个表主要是为了给不同用户上传的文件给予以不同的主键id值，而这样的主键id值在Django开发框架中的url模块中可以作为url地址中的参数从而跳转到不同的页面进行处理，从而实现了对于不同的文件有相对应的处理页面并且能够使上传了对应文件的用户能够成功下载得到对应训练数据的模型，避免了并发访问Web平台数据错乱的情况。

至于上传训练数据这一部分，我们使用了Django中提供的表单功能，我们的上传训练数据的表单类继承自Django中的forms.Form表单类，并且设置了widget参数，相对应的逻辑均可参照Django官方文档中的说明（对应的链接为[Django documentation | Django documentation | Django (djangoproject.com)](https://docs.djangoproject.com/en/4.2/)），并通过模板渲染出html页面，在对应的视图中设计上传训练数据文件的逻辑，通过捕获表单中的request为POST的方法利用对应Django中的文件表单的接口就可以实现将用户上传的文件保存到服务器本地。

除了需要上传训练数据集之外，还需要提供在线训练的功能，训练函数通过封装和Python中的模块与包的逻辑设计，将训练的逻辑封装成为一个函数并且导入到开发Web平台的服务器本地，并且在Web前端提供一个新的视图函数用来实现训练功能，在Django的子应用的url模块中添加对应的地址，并且在模板中渲染出对应的html页面，在视图函数中可以通过之前提到的url地址中的参数id来获取来到当前页面的对应的提交的训练数据的csv文件，通过数据库中存放的相关属性信息就可以在服务器中找到对应的文件并且调用训练函数进行训练即可。

最后一个功能是需要提供对训练得到的模型的下载功能，在封装好的训练函数中，训练完成后就会在指定的文件目录出生成对应的模型文件，因此这一个功能在Django中可以轻松实现，通过在对应的视图函数直接对对应的request请求返回一个FileResponse即可，这个FileResponse的参数就是调用Python中的open函数去打开指定的存放对应模型的目录地址即可。

### 2.3.2 预测模块

预测模块的要求是支持单条或者批量测试样本上传，并且可视化分类结果同时支持下载分类结果，这个功能的要求和训练模块基本上是一致的，即用户上传测试的数据文件，通过调用封装好的预测进行分类的函数，这个函数会使用训练生成的模型对上传的数据进行预测分类，并且会生成一个大赛要求的json格式的文件，提供对这个文件的下载即可，因此整个实现的逻辑和训练模块是如出一辙的，重复的逻辑就不再赘述了，几点实现中的注意点再说明一下。

首先，对于要求中的“支持单条或者批量测试样本上传”，一开始对于这样的要求是拟采用表单一项一项数据提交的方式，不过后来考虑到这样的方式是不妥的，要预测的数据可能是较为大量的，如果使用这样的方式对于测试样本的上传会变得极为繁琐，因此最终还是和训练模块中使用的上传训练数据文件一样的方式，统一采用csv格式文件上传的方式，对于单条或者批量的测试样本，可以统一放到对应的csv文件中然后多次上传即可，这样实现了接口的统一，并且极大地方便了测试样本的上传。

还有一点是要求中的“可视化分类结果”，这一点在实现中并没有考虑过多的设计，而是直接利用了在预测分类函数中为了生成要求的json格式的文件，会创建一个字典类型，通过直接在封装的函数返回这个字典，并将这个元素传参渲染到模板中的html中，通过简单设置相关的CSS格式，正确整齐地在html页面中输出每个故障预测的分类结果信息即可。

剩下的逻辑就和训练模块中基本一样了，不再赘述了。

### 2.3.3 整体部署

最后，在部署方面，出于方便访问的目的，我们希望自己开发的Web平台能够像平常在浏览器中访问的那些网址一样，所以我们自费在阿里云平台租界一台云服务器，并且将相关的环境都通过云服务器进行部署，这样在云服务器中启动Django项目中，在任何一台联网了的计算机的浏览器输入我们的云服务器的公网ip地址就可以进入到我们的Web平台进行使用了。

在有了公网访问的支持后，对于大赛要求的启动Web的可执行程序的提交也非常方便地完成了设计，首先创建一个python脚本文件，利用python的webbrowser库，可以通过这个python脚本文件去启动指定url地址，并且会自动打开浏览器进入对应的页面，进而通过pyinstaller将此python脚本打包成一个Windows平台下的可执行程序即可。

此外，在整体设计上完全遵循了前后端分离的设计模式，与机器学习有关的代码模块全部在后端实现并且部署，同时也对应提供了相关的接口，通过Django框架将前端和后端联系起来，这样在用户使用层面，很好地隔绝了用户和复杂的后端逻辑，既方便了用户使用他想用的功能，又很好地保护了相关数据资源，并且在设计过程中这样清晰的前后端分离的模式也极大地便利了我们的开发。

# 结果分析

## 3.1 模型性能评估

## 3.2 系统功能演示