|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | 成 绩 |  |
|  |  |  | 评阅人 |  |

**复 旦 大 学**

**研 究 生 课 程 论 文**

|  |  |
| --- | --- |
| 论文题目： | 基于分类算法的淡水质量预测 |
| 修读课程： | 机器学习（MSE620023） |
| 选课学期： | 2023春学期 |
| 选课学生： | 刘浩翔（22262010027） |
| 完成日期： | 2023/6/9 |

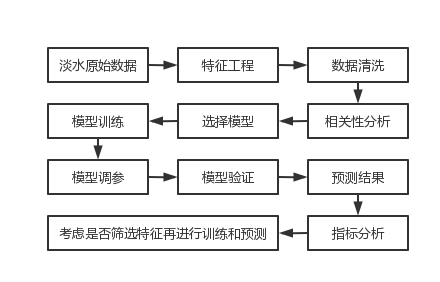
**基于分类算法的淡水质量预测**

本项目选取了2023春季英特尔oneAPI校园黑客松竞赛的“基于oneAPI的机器学习：预测淡水质量”题目。使用了英特尔oneAPI的AI分析工具包套和相应库（包括支持向量机SVM、随机森林、XGBoost），预测淡水是否可以安全饮用和被依赖淡水的生态系统所使用，从而可以帮助全球水安全和环境可持续性发展。

**1项目背景**

使用分类算法来进行水质预测，能够通过对已经经过专业检测进行判定水质数据进行学习来减少部分专业检测设备的检测成本，提高水质监测效率。通过对水质数据中的各种金属元素含量等的多维度智能分析，综合判断淡水 是否可以安全饮用和被依赖淡水的生态系统所使用。同时数据集中也采集了例如水温、气温等环境数据，同样也需要对这些外因进行分析，判断是否对淡水水质有影响。

**2淡水质量预测流程**

图2.1 淡水质量预测流程

如图2.1所示，本项目所设计的淡水质量预测的流程包括了如上11个步骤（实际操作的过程中更多）：淡水原始数据、特征工程、数据清洗、相关性分析、选择模型、模型训练、模型调参、模型验证、预测结果、指标分析和特征再筛选进行再训练。事实上，特征工程是将原始数据经过统计加工等方式变成新的样本特征。而数据清洗是将数据中的冗余值、缺失值乃至异常值等进行剔除，使数据集更加干净。相关性分析是在数据清洗和特征工程之后，把数据集各个维度的特征通过相关性分析来判断特征之间是否具有线性或者非线性关系，然后考虑是否需要进行特征的映射、变换或者剔除。然后本项目使用了多种模型，同时使用网格搜索（Grid Search）进行调参，使用训练好的模型进行预测。预测之后再考虑是否需要通过训练好的模型来选取更高重要性的特征，再进行上述步骤以提高模型预测的效果。

**3 数据集分析**

本次任务中，采用英特尔给出的淡水数据集，该数据集有六百万条以上的数据，每条数据中主要包括淡水质量是否合格以及水中检出的例如铁、硝酸盐、氯化物、铅、锌、氟化物、铜和硫酸盐等物质的含量，除此之外还有检测当天的气温、水温，水的ph、来源，导电率、总固体溶解量、日期等。直观来看，水中的各种物质含量会直接影响水的ph，但其他方面的影响并不直观，可能会存在潜在的非线性关系。

**3.1 数据预处理**

接下来我们要进行数据的预处理工作，首先需要导入Python的相关库，其中最重要的当然是Intel oneAPI的加速库：

import modin.pandas as pd  
from modin.config import Engine  
Engine.put("dask")

英特尔Modin分发版是一个分布式 DataFrame库，可根据数据集的大小无缝扩展pandas工作流，支持从1 MB到1TB以上的数据集。采用标准 pandas一次只能使用一个核心。然而，借助Modin Dask引擎，您可以使用所有可用核心，从而更快速地处理较大的数据集。

接下来是导入Intel Extension for Scikit-learn库，调用patch函数加速。

from sklearnex import patch\_sklearn  
patch\_sklearn()  
from sklearn.ensemble import RandomForestClassifier  
from sklearn import config\_context  
from sklearn.metrics import mean\_squared\_error, r2\_score, roc\_curve, precision\_recall\_curve, auc, make\_scorer, recall\_score, accuracy\_score, precision\_score, confusion\_matrix  
from sklearn.model\_selection import train\_test\_split, GridSearchCV, StratifiedKFold  
import sklearn.linear\_model as lm  
import time  
import warnings  
import pandas  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt  
import matplotlib.colors  
import plotly.io as pio  
import plotly.express as px  
import plotly.graph\_objects as go  
from plotly.subplots import make\_subplots

接下来，我们使用Modin Dask引擎支持的pandas进行数据的读取数据，然后查看原始数据集的维度。然后使用pandas自带的方法dataframe.info()进行数据的信息概览查看。最后使用dataframe.isna()、dataframe..dropna()进行缺失值的查看和删除。其中scikit-learn我们使用了Intel oneAPI中可以调用patch函数加速的版本。其中sklearn.model\_selection中的train\_test\_split，GridSearchCV，StratifiedKFold分别用于分割训练、测试集，使用网格搜索，交叉验证。numpy是科学计算库，十分方便对矩阵和高维向量进行处理和计算，广泛应用在机器学习和深度学习领域中。matplotlib是对数据进行可视化分析的常见工具之一。

**3.2 数据探查与清洗**

首先我们需要用modin.pandas将数据集读取，转化成DataFrame对象，并将样本数量、字段数量、数据类型、非空值数量等详细信息进行显示。

data = pd.read\_csv('.\dataset\dataset.csv')  
print("Data shape: {}\n".format(data.shape))  
display(data.head())

data.info()

运行后，其结果输出如下：

<class 'modin.pandas.dataframe.DataFrame'>

RangeIndex: 5956842 entries, 0 to 5956841

Data columns (total 24 columns):

# Column Non-Null Count Dtype

--- ---------------------- ---------------- -----

0 Index 5956842 non-null int64

1 pH 5840788 non-null float64

2 Iron 5917089 non-null float64

3 Nitrate 5851117 non-null float64

4 Chloride 5781311 non-null float64

5 Lead 5929933 non-null float64

6 Zinc 5800716 non-null float64

7 Color 5951103 non-null object

8 Turbidity 5907027 non-null float64

9 Fluoride 5767686 non-null float64

10 Copper 5757440 non-null float64

11 Odor 5777951 non-null float64

12 Sulfate 5759424 non-null float64

13 Conductivity 5792981 non-null float64

14 Chlorine 5899017 non-null float64

15 Manganese 5847259 non-null float64

16 Total Dissolved Solids 5955172 non-null float64

17 Source 5868580 non-null object

18 Water Temperature 5788609 non-null float64

19 Air Temperature 5927114 non-null float64

20 Month 5861174 non-null object

21 Day 5857239 non-null float64

22 Time of Day 5842323 non-null float64

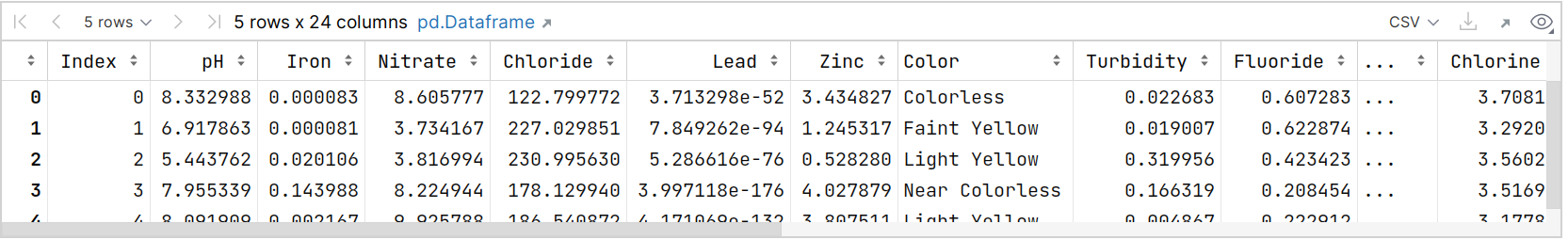
23 Target 5956842 non-null int64

dtypes: float64(19), object(3), int64(2)

memory usage: 1.1 GB

可以看到一共有5956842条数据，共有24列，共占用内存1.1GB。

同时通过display得到的数据DataFrame预览如下：

图3.1 DataFrame预览

然后我们使用dataframe.isna()来查看缺失的数据，并进行清洗，代码如下：

display(data.isna().sum())  
missing=data.isna().sum().sum()  
duplicates=data.duplicated().sum()  
print("\nThere are {:,.0f} missing values in the data.".format(missing))  
print("There are {:,.0f} duplicate records in the data.".format(duplicates))

等到结果如下：

Index 0

pH 116054

Iron 39753

Nitrate 105725

Chloride 175531

Lead 26909

Zinc 156126

Color 5739

Turbidity 49815

Fluoride 189156

Copper 199402

Odor 178891

Sulfate 197418

Conductivity 163861

Chlorine 57825

Manganese 109583

Total Dissolved Solids 1670

Source 88262

Water Temperature 168233

Air Temperature 29728

Month 95668

Day 99603

Time of Day 114519

Target 0

dtype: int64

There are 2,369,471 missing values in the data.

There are 0 duplicate records in the data.

可以看到，除了Index和Target没有缺失数据以外，其他数据均还有一定规模的缺失条目。我们使用dataframe.dropna()对数据进行清洗。代码如下：

data = data.dropna()  
display(data.isna().sum())  
missing=data.isna().sum().sum()  
duplicates=data.duplicated().sum()  
print("\nThere are {:,.0f} missing values in the data.".format(missing))  
print("There are {:,.0f} duplicate records in the data.".format(duplicates))

等到如下图结果：

Index 0

pH 0

Iron 0

Nitrate 0

Chloride 0

Lead 0

Zinc 0

Color 0

Turbidity 0

Fluoride 0

Copper 0

Odor 0

Sulfate 0

Conductivity 0

Chlorine 0

Manganese 0

Total Dissolved Solids 0

Source 0

Water Temperature 0

Air Temperature 0

Month 0

Day 0

Time of Day 0

Target 0

dtype: int64

There are 0 missing values in the data.

There are 0 duplicate records in the data.

由此可知，数据已经清洗完毕。

**3.3 数据特征初步筛选**

通过上述分析，我们其实可以看到有一部分数据是用Object格式表示，不利于我们的模型进行训练，我们可以采用将它抽象化或者其他方式进行处理。本项目直接移除了month, day, time of day三个用于标记时间戳的无效字段, 和color, source两个string类型字段。使用如下代码：

data = data[['Index', 'pH', 'Iron', 'Nitrate', 'Chloride', 'Lead', 'Zinc', 'Turbidity', 'Fluoride', 'Copper', 'Odor', 'Sulfate', 'Conductivity', 'Chlorine', 'Manganese', 'Total Dissolved Solids', 'Water Temperature', 'Air Temperature', 'Target']]  
print(data.columns.tolist())  
print("Data shape: {}\n".format(data.shape))  
代码运行结果如下：

['Index', 'pH', 'Iron', 'Nitrate', 'Chloride', 'Lead', 'Zinc', 'Turbidity', 'Fluoride', 'Copper', 'Odor', 'Sulfate', 'Conductivity', 'Chlorine', 'Manganese', 'Total Dissolved Solids', 'Water Temperature', 'Air Temperature', 'Target']

Data shape: (3981800, 19)

我们可以知道清洗再进行特征筛选后仅剩下3981800条数据，且数据有19列。同时我们再对数据集的Target字段（即样本标签）进行输出，使用如下代码：

data['Target'].value\_counts()

得到输出：

0 2774344

1 1207456

Name: Target, dtype: int64

数据集没有具体说明哪个类别是代表淡水质量可用，我们暂且认为是类别1。同时观察到样本分布在400万的数据量级下还是比较均衡的（个人认为），所以不进行不平衡样本处理操作（例如分层抽样等）。

**3.4 数据相关性分析**

我们需要对数据样本进行一定的相关性分析，来探查数据集各个特征之间的关系，同时为我们后续的特征筛选以及模型算法的选取做出指导性的建议。我们先采用概率密度函数直观显示数据形态。代码如下：

fig, ax = plt.subplots(9,2, figsize=(16,18))

fig.suptitle('Distribution of Numerical Variables',fontsize=12)

row=0

col=[0,1]\*9

for i, column in enumerate(plot\_df.columns[1:]):

print(i, column)

if (i!=0) & (i%2==0):

row+=1

sns.kdeplot(x=column, hue='Target', palette=intel\_pal[::-1], hue\_order=[1,0],

label=['State 1','State 0'], data=plot\_df,

fill=True, linewidth=2.5, legend=False, ax=ax[row,col[i]])

ax[row,col[i]].tick\_params(left=False, bottom=False)

ax[row,col[i]].set(title='\n\n{}'.format(column), xlabel='', ylabel=('Density' if i%2==0 else ''))

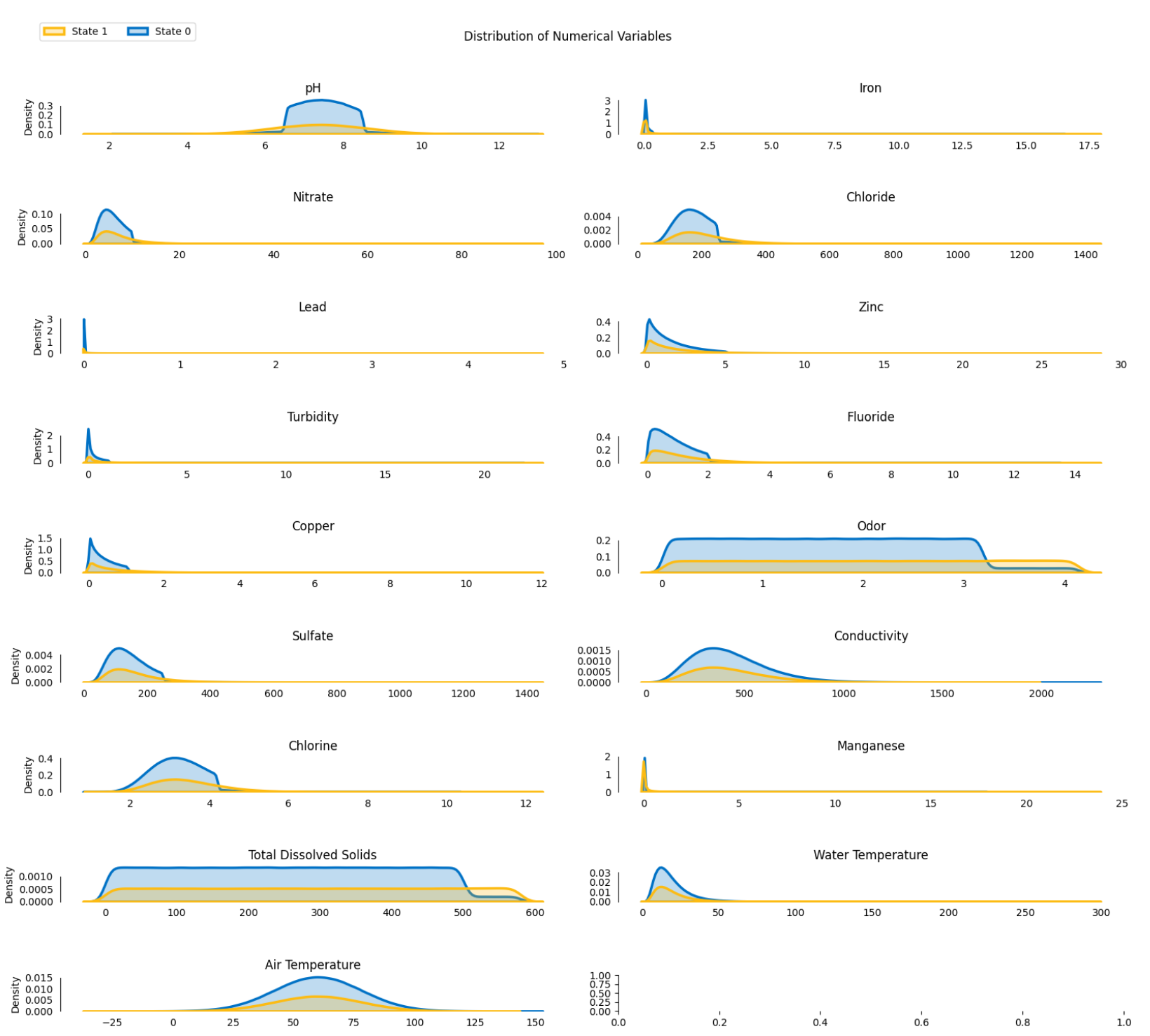
handles, \_ = ax[0,0].get\_legend\_handles\_labels()

fig.legend(labels=['State 1','State 0'], handles=reversed(handles), ncol=2, bbox\_to\_anchor=(0.18, 0.99))

sns.despine(bottom=True, trim=True)

plt.tight\_layout(rect=[0, 0.2, 1, 0.99])

输出的结果如下图所示：

图3.2 一元分布图

图中结果反应出除了Odor、Chlorine呈现出稳定的均匀且非正态分布，其他特征都呈现一定的正态分布特征。

通过上述观察，其实不难发现，我们数据的特征和目标之间存在着非线性的关系，无法使用参数模型，需要尝试使用非参数模型，包括支持向量机（SVM）、随机森林、XGBoost等。与此同时我们可以使用散点图来描述一个从ph到各个其他特征之间的关系，代码如下：

plot\_df=data[float\_cols]

fig, ax = plt.subplots(5,4, figsize=(16,18))

fig.suptitle('Scatterplot Matrix of Numeric Variables\nLog-Transformed',fontsize=12)

print(float\_cols[1])

i = 0

col = float\_cols[1]

for j, iter\_col in enumerate(float\_cols[0:]):

if j < 4:

k = j

else:

k = j % 4

if j % 4 == 0:

i += 1

print('j:',j,'iter\_col:' ,iter\_col,'k:',k)

ax[i,k].hexbin(x=iter\_col, y=col, data=plot\_df, bins='log', gridsize=40, cmap='coolwarm')

ax[i,k].set(xlabel=iter\_col, ylabel= col) #(col if j%3==0 else ''))

ax[i,k].tick\_params(left=False,bottom=False)

sns.despine()

plt.tight\_layout(rect=[0, 0, 1, 0.99])

plt.show()

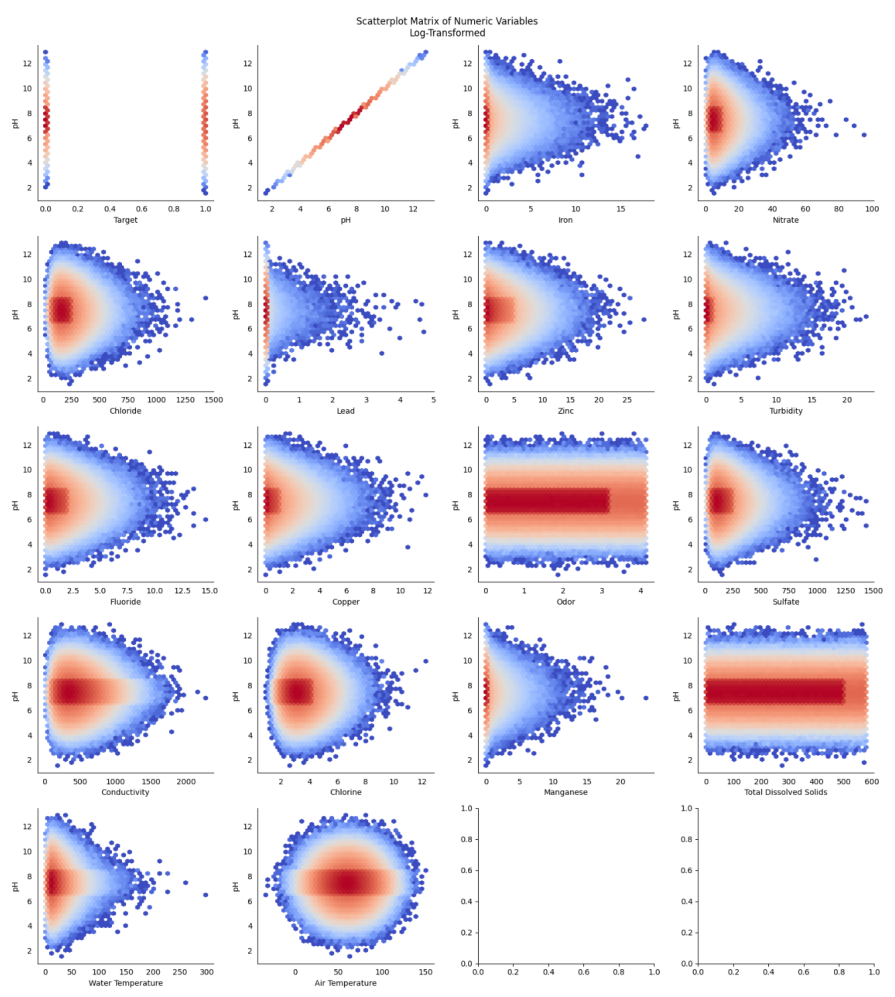
得到如下图的散点图：

图3.3 关于ph的散点图

散点图中我们可以更加明显的看出，数据主要集中在ph6到8之间，且看不出直观的线性关系，更加印证了先前我们认为的特征之间呈现非线性关系，需要使用非参数的模型进行训练和预测的观点。

上述使用数值的分布来表现特征之间的关系，接下来我们直接绘制每个特征之间的相关性矩阵，使用代码如下：

def plot\_corr(corr):

"""

Function to plot bottom left triangle of correlation matrix

"""

mask=np.triu(np.ones\_like(corr, dtype=bool))[1:,:-1]

corr=corr.iloc[1:,:-1].copy()

fig, ax = plt.subplots(figsize=(26,22))

sns.heatmap(corr, mask=mask, vmin=-1, vmax=1, center=0, annot=True, fmt='.2f', cmap='YlGnBu\_r',lw=2, annot\_kws={'fontsize':10,'fontweight':'bold'}, cbar=True)

ax.tick\_params(left=False,bottom=False)

ax.set\_xticklabels(ax.get\_xticklabels(), rotation=45, horizontalalignment='right',fontsize=12)

ax.set\_yticklabels(ax.get\_yticklabels(),fontsize=12)

plt.title('Correlations between Utility Asset Maintenance Data\n', fontsize=24)

plt.show()

def plot\_target\_corr(corr, target\_col):

"""

Function to plot a bar chart of correlations between target and features, sorted in descending order

"""

corr=corr[target\_col].sort\_values(ascending=False)[1:]

pal=sns.color\_palette("RdYlBu",37).as\_hex()

pal=[j for i,j in enumerate(pal) if i not in (17,18)]

rgb=['rgba'+str(matplotlib.colors.to\_rgba(i,0.8)) for i in pal]

fig=go.Figure()

fig.add\_trace(go.Bar(x=corr.index, y=corr, marker\_color=rgb,

marker\_line=dict(color=pal,width=2),

hovertemplate='%{x} correlation with Target = %{y}',

showlegend=False, name=''))

fig.update\_layout(template=temp, title='Feature Correlations with Target (Asset Label)',

yaxis\_title='Correlation', margin=dict(b=160), xaxis\_tickangle=45)

fig.show()

corr=data.corr()

plot\_corr(corr=corr)

plot\_target\_corr(corr=corr, target\_col='Target')

得到如下的结果图：

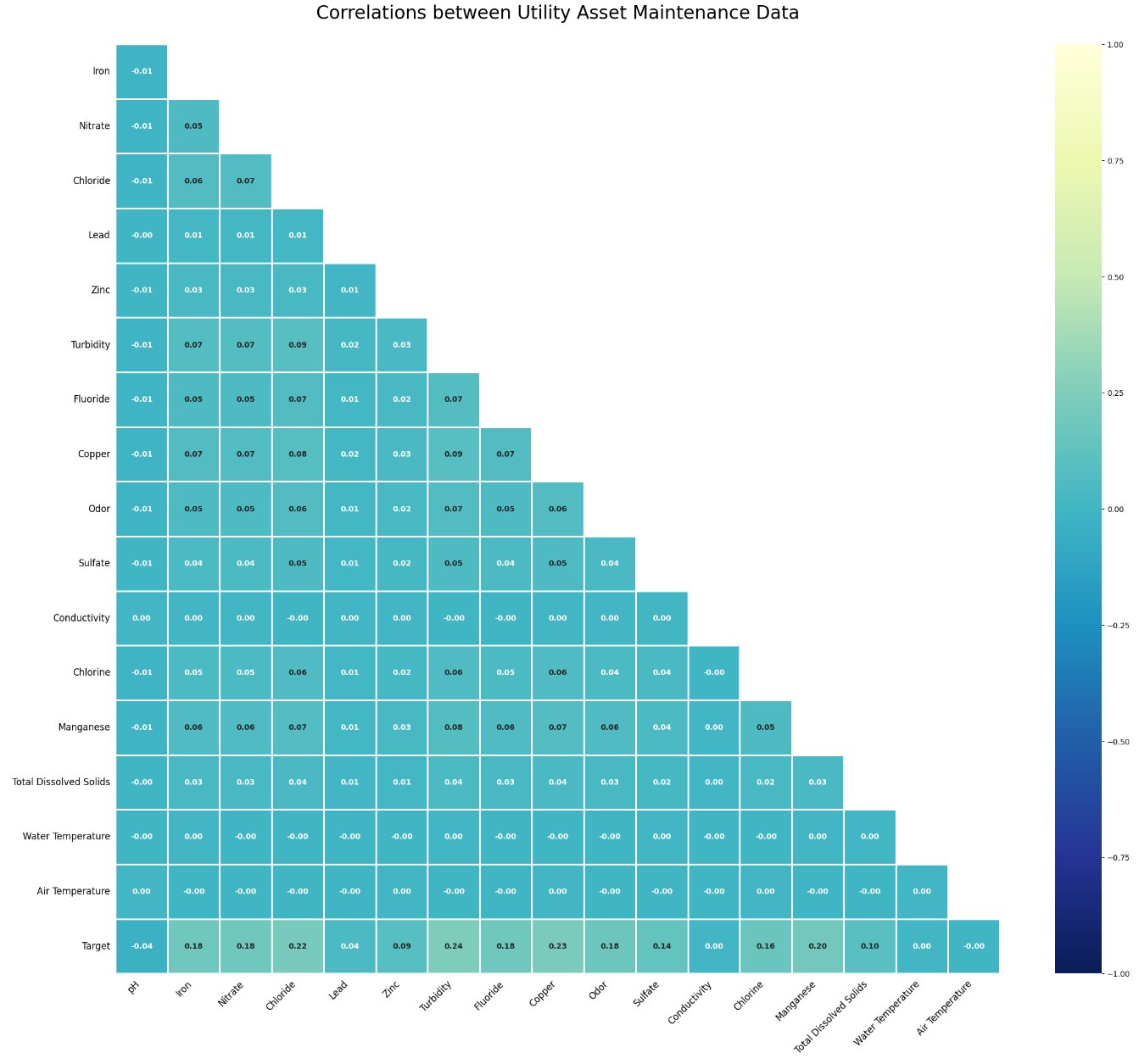


图3.4 各特征之间的相关性

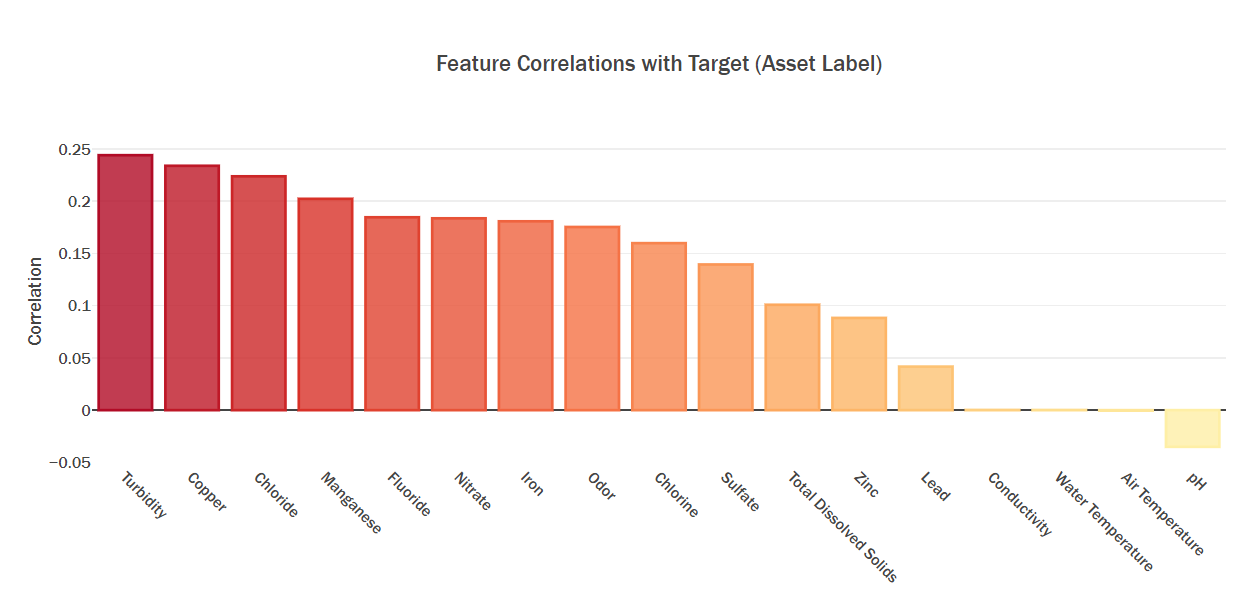


图3.5 各特征与目标的相关性（从大到小）

由图3.4可知单独的特征与特征之间，相关型非常低，所以可以暂时不需要对某个特征进行一些映射处理消除相关性。同时从图3.5可知Turbidity、Copper与目标Target呈最高的正相关性，pH和目标Target呈最高的负相关性，而Conductivity、Water Temperature、Air Temperature与Target基本没有相关性，我们暂且认为可以将他们从特征中剔除也不会对结果产生很大的影响，且可以提高训练和预测速度。

**3.5数据集划分与标准化处理**

为了之后的模型训练，我们需要把数据集整体进行划分，这里我选取了7：3的划分方法，将数据集划分为训练集和测试集两部分。另外，对所有输入变量采用sklearn中的StandardScaler标准化方法、RobustScaler方法将数据的输入范围映射到更合理的区间上。我们需要导入一些另外的Python库，如下所示：

from sklearn.preprocessing import RobustScaler, MinMaxScaler

from sklearn.metrics import roc\_auc\_score, roc\_curve, auc, accuracy\_score, f1\_score

from sklearn.metrics import precision\_recall\_curve, average\_precision\_score

from sklearn.svm import SVC

import os  
import daal4py as d4p  
from xgboost import XGBClassifier

同时我们还将下一步需要的SVM模型、XGBoost模型也一并进行了导入。其中Daal4py可从 英特尔oneAPI数据分析库(oneDAL) 中下载，它将利用底层的英特尔的高级矢量扩展指令集（英特尔AVX-512）硬件，最大限度地提高英特尔至强处理器上的梯度提升性能。代码实例如下图，由于对不同模型做了不同的处理，此处只贴一份代码：

def prepare\_train\_test\_data(data, target\_col, test\_size):

"""

Function to scale and split the data into training and test sets

"""

scaler = RobustScaler()

X = data.drop(target\_col, axis=1)

# 'Water Temperature', 'Air Temperature'

# X = X.drop('Water Temperature', axis=1)

# X = X.drop('Air Temperature', axis=1)

y = data[target\_col]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=test\_size, random\_state=21)

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

print("Train Shape: {}".format(X\_train\_scaled.shape))

print("Test Shape: {}".format(X\_test\_scaled.shape))

return X\_train\_scaled, X\_test\_scaled, y\_train, y\_test

print("Preparing Train and Test datasets")

X\_train, X\_test, y\_train, y\_test = prepare\_train\_test\_data(data=data,

target\_col='Target',

test\_size=.25)

预处理后得到训练集和测试集：

Preparing Train and Test datasets

Train Shape: (2986350, 17)

Test Shape: (995450, 17)

**4 模型选择**

**4.1 支持向量机**

根据课堂所学和经验可知，支持向量机只适用于少部分样本且维度不能过高的情况，否则支持向量机难以使用一个超平面将数据集有效的划分为两个类别。本项目的初期尝试使用SVM在该数据集上进行训练，实际结果差强人意，甚至在400万的完备数据集下一天一夜也没有训练出结果，故而分别取训练集和数据集的前100000个数据进行训练和测试。代码如下所示：

X\_train\_10000, y\_train\_10000, X\_test\_10000, y\_test\_10000 = X\_train[:100000], y\_train[:100000], X\_test[:100000], y\_test[:100000]

## Initialize SVC model ##

parameters = {

'class\_weight': 'balanced',

'probability': True,

'random\_state': 21

}

svc = SVC(\*\*parameters)

strat\_kfold = StratifiedKFold(n\_splits=3, shuffle=True, random\_state=21)

grid = {

'C': np.logspace(-1, 1, 5),

'kernel': ['linear', 'poly', 'rbf', 'sigmoid']

}

grid\_search = RandomizedSearchCV(svc, param\_distributions=grid, cv=strat\_kfold, n\_iter=5, scoring='roc\_auc', verbose=1, n\_jobs=-1, random\_state=21)

grid\_search.fit(X\_train\_10000, y\_train\_10000)

print("Trainning finished, best hyperparameters: ", grid\_search.best\_params\_)

print("Best cross-validation AUC: {:.4f}".format(grid\_search.best\_score\_))

svc = grid\_search.best\_estimator\_

svc\_prob = svc.predict\_proba(X\_test\_10000)[:, 1]

svc\_pred = pd.Series(svc.predict(X\_test\_10000), name='Target')

svc\_auc = roc\_auc\_score(y\_test\_10000, svc\_prob)

svc\_f1 = f1\_score(y\_test\_10000, svc\_pred)

## Print model results ##

print("\nTest F1 accuracy: {:.2f}%, AUC: {:.5f}".format(svc\_f1\*100,svc\_auc))

最后输出结果如下：

Fitting 3 folds for each of 5 candidates, totalling 15 fits

Trainning finished, best hyperparameters: {'kernel': 'rbf', 'C': 1.0}

Best cross-validation AUC: 0.9031

Test F1 accuracy: 77.00%, AUC: 0.90179

我们可以看到AUC指标为0.9031，测试集上的F1指标为0.77，效果虽然不是特别好，但是我们还是可以通过此得出结论，该数据集是线性可分的，理论上通过合理的参数，在更大量样本的数据集中，SVM仍然有效，且可预知的是使用其他更合理的模型可以获得比SVM更好的效果。同时本项目也做了一次使用少量训练集进行训练，然后在大量样本的测试集上进行验证的事情，具体的，我们将上文代码中的测试集换成了初步数据集分割后的含有100万左右数据的测试集进行测试，得到如下结果：

Fitting 3 folds for each of 5 candidates, totalling 15 fits

Trainning finished, best hyperparameters: {'kernel': 'rbf', 'C': 1000.0}

Best cross-validation AUC: 0.8259

Test F1 accuracy: 64.93%, AUC: 0.81669

由F1和AUC指标可以了解到，模型的总体预测能力尚可，但是在某个类别上的分类能力较弱，推测可知，模型在1样本（不可用水质）上存在较多的误判。接下来我们使用其他模型来进行预测。

**4.2 XGBoost**

针对SVM模型无法对全样本进行训练的情况，我决定采用集成方法，XGBoost进行训练和预测。与SVM相同，我们先对少量样本进行分割训练先得出一个快速的结论（取前10万样本中的7万作为训练集，3万作为测试集）。代码如下：

## Prepare Train and Test datasets ##

print("Preparing Train and Test datasets")

X\_train, X\_test, y\_train, y\_test = prepare\_train\_test\_data(data=data,

target\_col='Target',

test\_size=.25)

X\_train, X\_test, y\_train, y\_test = X\_train[:70000], X\_test[:30000], y\_train[:70000], y\_test[:30000]

## Initialize XGBoost model ##

ratio = float(np.sum(y\_train == 0)) / np.sum(y\_train == 1)

parameters = {'scale\_pos\_weight': ratio.round(2), # 通过采样方法解决样本不平衡问题

'random\_state': 21,

'lambda': 2}

xgb\_model = XGBClassifier(\*\*parameters)

## Tune hyperparameters ##

strat\_kfold = StratifiedKFold(n\_splits=3, shuffle=True, random\_state=21)

print("\nTuning hyperparameters..")

grid = {'min\_child\_weight': [15, 25],

'gamma': [1, 5,10, 15],

'max\_depth': [5, 10, 15, 20],

'tree\_method': ['hist', 'approx'],

}

grid\_search = GridSearchCV(xgb\_model, param\_grid=grid,

cv=strat\_kfold, scoring='roc\_auc',

verbose=1, n\_jobs=-1)

grid\_search.fit(X\_train, y\_train)

print("Done!\nBest hyperparameters:", grid\_search.best\_params\_)

print("Best cross-validation AUC: {:.4f}".format(grid\_search.best\_score\_))

## Convert XGB model to daal4py ##

xgb = grid\_search.best\_estimator\_

daal\_model = d4p.get\_gbt\_model\_from\_xgboost(xgb.get\_booster())

## Calculate predictions ##

daal\_prob = d4p.gbt\_classification\_prediction(nClasses=2,

resultsToEvaluate="computeClassLabels|computeClassProbabilities",

fptype='float').compute(X\_test, daal\_model).probabilities # or .predictions

xgb\_pred = pd.Series(np.where(daal\_prob[:,1]>.5, 1, 0), name='Target')

xgb\_auc = roc\_auc\_score(y\_test, daal\_prob[:,1])

xgb\_f1 = f1\_score(y\_test, xgb\_pred)

## Plot model results ##

print("\nTest F1 Accuracy: {:.2f}%, AUC: {:.5f}".format(xgb\_f1\*100, xgb\_auc))

plot\_model\_res(model\_name='XGBoost', y\_test=y\_test, y\_prob=daal\_prob[:,1])

得到的结果如下与下图所示：

Tuning hyperparameters..

Fitting 3 folds for each of 64 candidates, totalling 192 fits

Done!

Best hyperparameters: {'gamma': 10, 'max\_depth': 10, 'min\_child\_weight': 15, 'tree\_method': 'approx'}

Best cross-validation AUC: 0.9123

Test F1 Accuracy: 83.29%, AUC: 0.90805

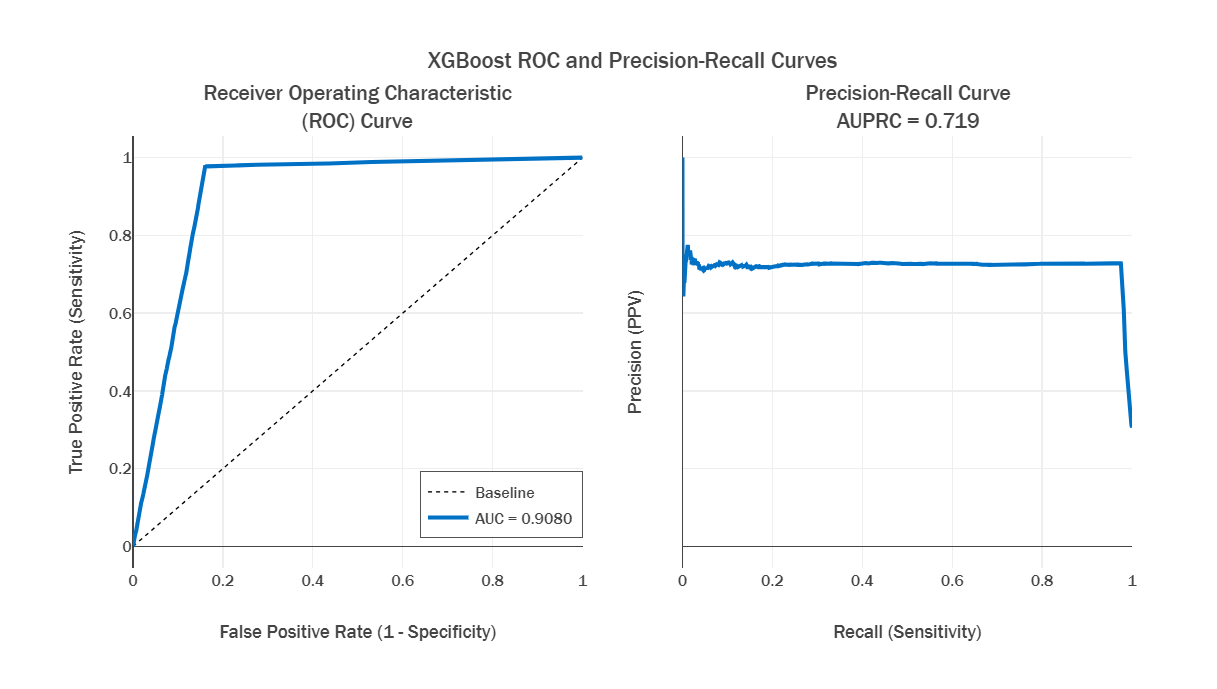


图4.1 10万数据量XGBoost上的ROC和AUPRC曲线

我们通过xgboost中的feature\_importances\_函数按照模型计算出的特征值的重要性进行特征的再次筛选然后再训练，以求得到更好的预测结果。使用如下代码，筛去重要性小于0.01的特征：

feature\_arr = xgb.feature\_importances\_ > 0.01

result\_f\_arr = []

for i, label in enumerate(feature\_arr):

if label:

result\_f\_arr.append(data.columns[i])

result\_f\_arr.append('Target')

print(result\_f\_arr)

得到结果：

['pH', 'Iron', 'Nitrate', 'Chloride', 'Zinc', 'Turbidity', 'Fluoride', 'Copper', 'Odor', 'Sulfate', 'Chlorine', 'Manganese', 'Total Dissolved Solids', 'Target']

除去Target，特征还剩下13个维度，我们使用这13个维度进行再次训练和预测（此处代码与上无异故不贴出代码），得到如下结果：

Tuning hyperparameters..

Fitting 3 folds for each of 64 candidates, totalling 192 fits

Done!

Best hyperparameters: {'gamma': 1, 'max\_depth': 20, 'min\_child\_weight': 15, 'tree\_method': 'approx'}

Best cross-validation AUC: 0.9131

Test F1 Accuracy: 83.14%, AUC: 0.91497

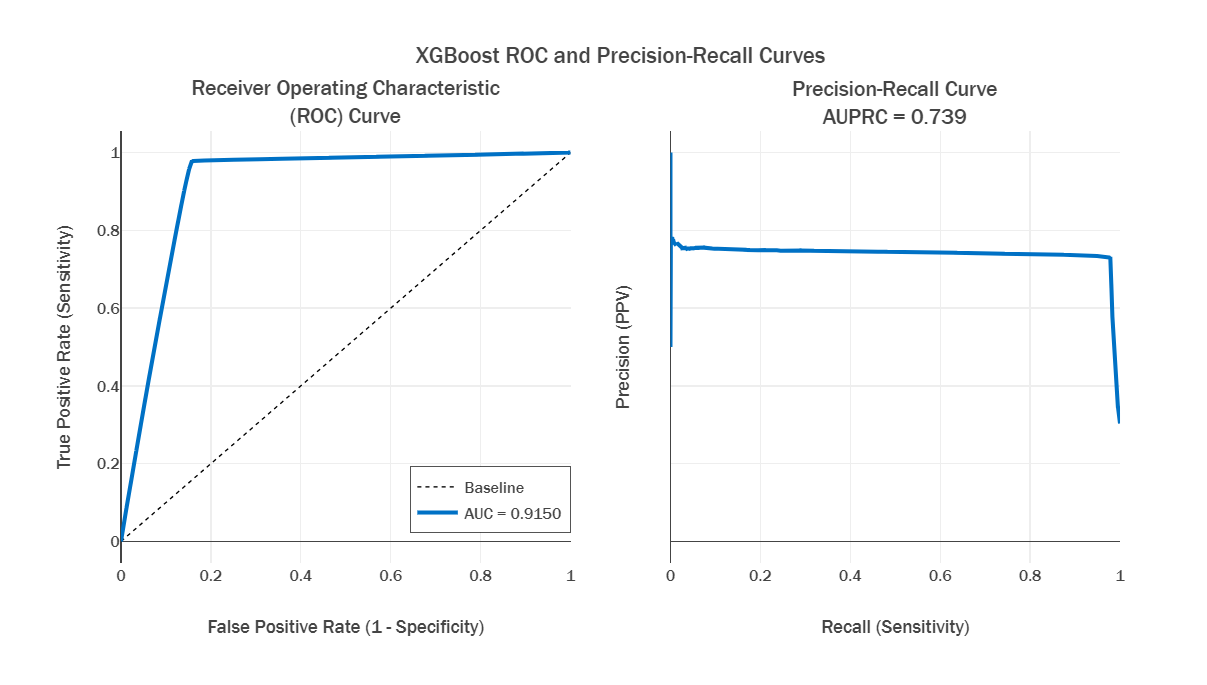


图4.2 XGBoost全数据集结果

由此可见AUC指标尚可，AUPRC指标有了微弱的提升，F1指标基本持平，根据经验和知识分析，确实是由样本分布不均衡导致，故而可能只能采取上采样或者下采样等方式，进行数据集的平衡。

**4.3 XGBoost优化**

上文中，我们针对XGBoost已经做了利用网格搜索（Grid Search）的优化，筛选特征值的优化，但效果均不显著，尤其是在AUPRC和F1指标上，故而样本平衡问题亟待解决。我们采用上采样的方式，对偏少量的样本进行过采样，再进行训练，代码如下：

from imblearn.over\_sampling import RandomOverSampler

def prepare\_train\_test\_data(data, target\_col, test\_size):

"""

Function to scale and split the data into training and test sets

"""

ros = RandomOverSampler(random\_state=21)

scaler = RobustScaler()

X = data.drop(target\_col, axis=1)

# 'Water Temperature', 'Air Temperature'

# X = X.drop('Water Temperature', axis=1)

# X = X.drop('Air Temperature', axis=1)

y = data[target\_col]

X, y = ros.fit\_resample(X, y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=test\_size, random\_state=21)

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

print("Train Shape: {}".format(X\_train\_scaled.shape))

print("Test Shape: {}".format(X\_test\_scaled.shape))

return X\_train\_scaled, X\_test\_scaled, y\_train, y\_test

## Prepare Train and Test datasets ##

print("Preparing Train and Test datasets")

X\_train, X\_test, y\_train, y\_test = prepare\_train\_test\_data(data=data\_res,

target\_col='Target',

test\_size=.25)

利用处理过后的数据集训练和预测，得到如下结果：

Tuning hyperparameters..

Fitting 3 folds for each of 32 candidates, totalling 96 fits

Done!

Best hyperparameters: {'gamma': 1, 'max\_depth': 20, 'min\_child\_weight': 15}

Best cross-validation AUC: 0.9313

Test F1 Accuracy: 91.54%, AUC: 0.93875

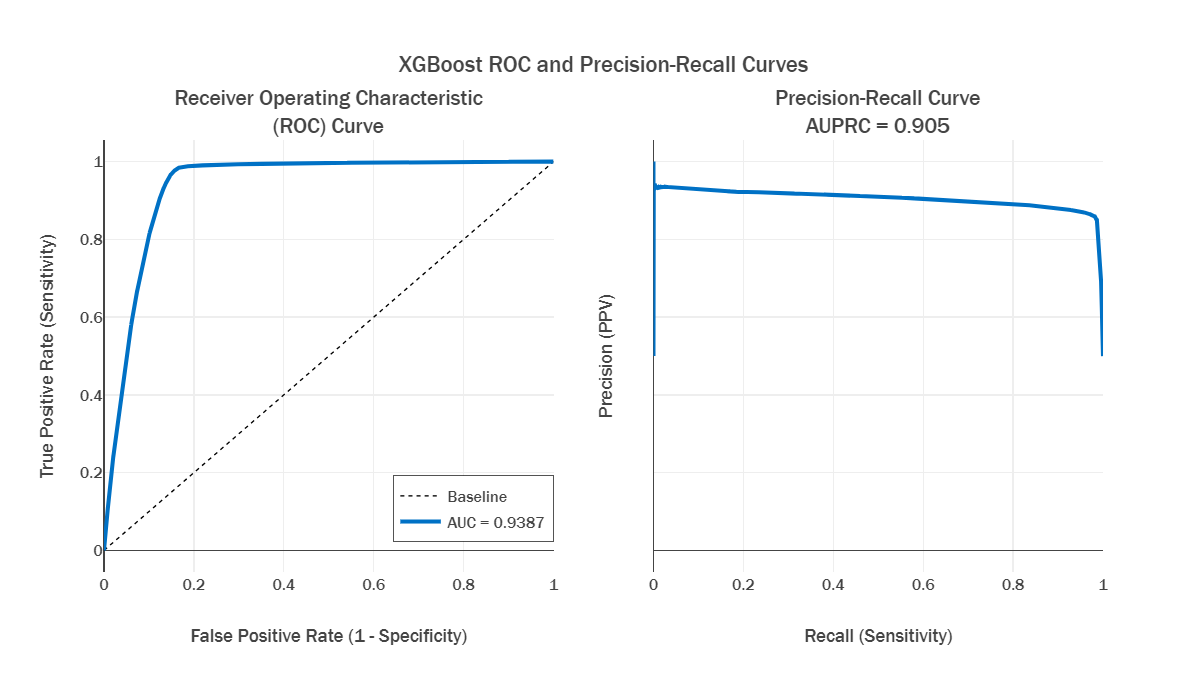


图4.3 XGBoost上采样全数据集结果

我们惊人的发现，在过采样后，效果得到了显著的提升，尤其是在AUPRC和F1指标上，由此可见，样本不平衡会对这两个指标产生较大的影响。