第5章 预测淡水质量

地球上的淡水仅占总水量的3%，它是人类最重要的自然资源之一。淡水几乎触及人类日常生活的方方面面，从饮用、游泳、沐浴到生产食物和每天使用的产品。获得安全卫生的供水不仅对人类生活至关重要，而且对遭受干旱、污染和气温升高影响的周边生态系统的生存也至关重要。

在稀缺的淡水资源中，可供日常饮用的部分还需要经过复杂的处理流程。未经处理的淡水资源中含有大量的细菌、病毒、寄生虫、重金属等物质，它们会对人体健康造成巨大的危害。此外，淡水中还可能含有一些化学物质，例如氟化物、硝酸盐等。因此，预测淡水是否可以安全饮用，对帮助全球水安全和环境可持续性发展尤为重要。

5.1 数据清洗处理

可饮用淡水的质量指标包括很多种类。如表5.1所示，pH是度量水的酸碱度的指标，浑浊度是度量水中悬浮物质的指标，氨氮、总磷、总氮等是衡量水中富营养化的指标，重金属则是度量水中污染物的指标。此外，还有水源、水的颜色、气味以及水温等指标。在数据处理中，数据集中的索引、气温、月、日、时间等属性对预测没有作用，可以忽略。

表5.1 关键属性表

关 键 词	含 义	选 用 理 由
pH	溶液酸碱度	酸碱度是重要标准
Iron	铁元素	重金属元素，超标会导致肠道不适
Nitrate	硝酸盐	超标会降低血液运氧能力
Chloride	氯化物	富营养化
Lead	铅元素	重金属元素，超标危害健康
Zinc	锌元素	重金属元素，超标危害健康
Color	颜色	绿色往往富营养化，有机物高
Turbidity	浑浊度	微生物相关指标
Fluoride	氟化物	富营养化指标
Copper	铜元素	重金属元素，超标危害健康
Odor	气味	侧面反映细菌微生物状况
Sulfate	硫酸盐	超标会导致肠道不适
Conductivity	电导率	反映溶解盐含量
Chlorine	氯元素	与其化合物为主要富营养化
Manganese	锰元素	超标慢性中毒，危害神经系统
Total Dissolved Solids	总溶解固体	溶解的无机盐有机物总量

续表

关 键 词	含 义	选 用 理 由
Source	水源	淡水的来源
Water Temperature	水温	影响含氧量微生物

除选取关键的特征外,还需要对数据中的噪声、缺失值、异常值进行处理,这些问题数据可能是由采集数据的漏洞所导致的,会降低模型训练的精确度。

5.1.1 缺失值的处理

数据中的缺失值会随模型的训练而产生负面影响,由于很多机器学习的分类算法不支持数据中的缺失值,因此需要对缺失值进行处理。首先使用 Python 中的 MissingNo 库可视化地展示原始数据中缺失值的比例。这样既可以快速直观地了解数据的完整性,又能将数据的缺失值记录与比例按照矩阵形式演示。如图 5.1 所示,原始数据中几乎各个特征都包含一定的缺失值。直接删除这些样本的缺失值是简单快速的处理方式,但是可能会丢失隐藏的知识点,影响模型的训练精度和泛化能力。

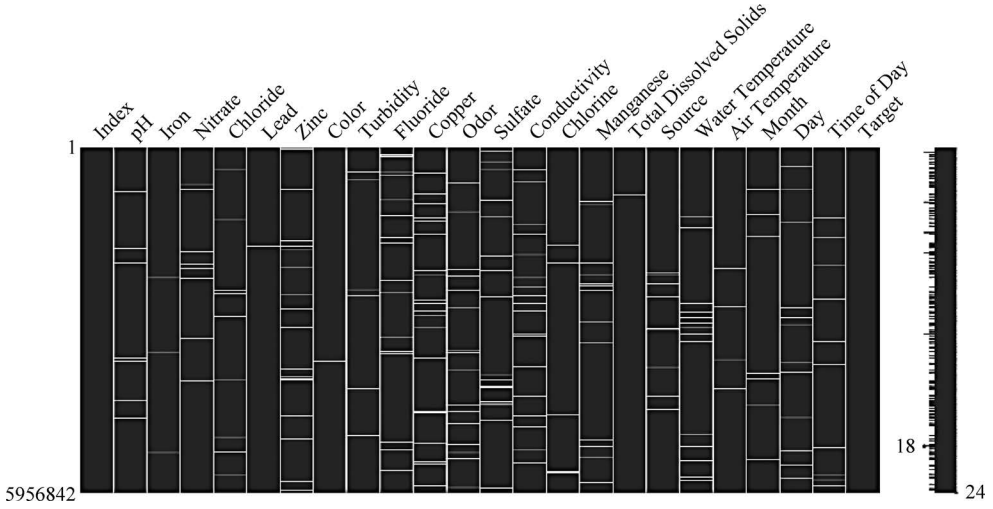

图 5.1 样本数据缺失矩阵

在包含缺失值的原始数据中,来源(Source)与颜色(Color)是类别属性的原始数据,其余的特征都是数值类的数据。类别属性有两种处理方式:一种是使用出现最多次数的标签填充;另一种是把缺失的标签单独归为一类进行分析。数值型的数据则可以使用取值最多的数或者平均数填充。将数值型的特征名称保存在 param_distribute_list 的变量名中,使用取值最多的数进行填充。

```
for column in param_distribute_list:
most_frequent_value = df[column].mode().values[0]
df[column].fillna(most_frequent_value, inplace = True)
print(column,'fillsuccessed')
```

5.1.2 特征数值分布

在机器学习中,各个特征的质量分布可以帮助了解它们的分布情况以及特征之间的相

关性，这有助于选择合适的特征或者进行特征变换，以便更好地挖掘数据的内在规律。为了便于总览数据以及节省空间，可以将 16 个特征分布图直接绘制成 4×4 的紧凑格式。

```
def plot_distribute(df,titles):
    fig,axs = plt.subplots(nrows = 4,ncols = 4,figsize = (12,12))
    for i in range(16):
        title = titles[i]
        row = i//4
        col = i % 4
        ax = axs[row][col]
        data = df[title]
        ax.hist(data,bins = 50)
        ax.set_title(title)
    plt.tight_layout()
    plt.show()
```

如图 5.2 所示的质量分布图，pH、硝酸盐、氯化物、硫酸盐、水温属于类正态分布，多数重金属元素的质量分布更偏向于 0 值的偏态分布，其中一个可能的原因是数据需要进行转

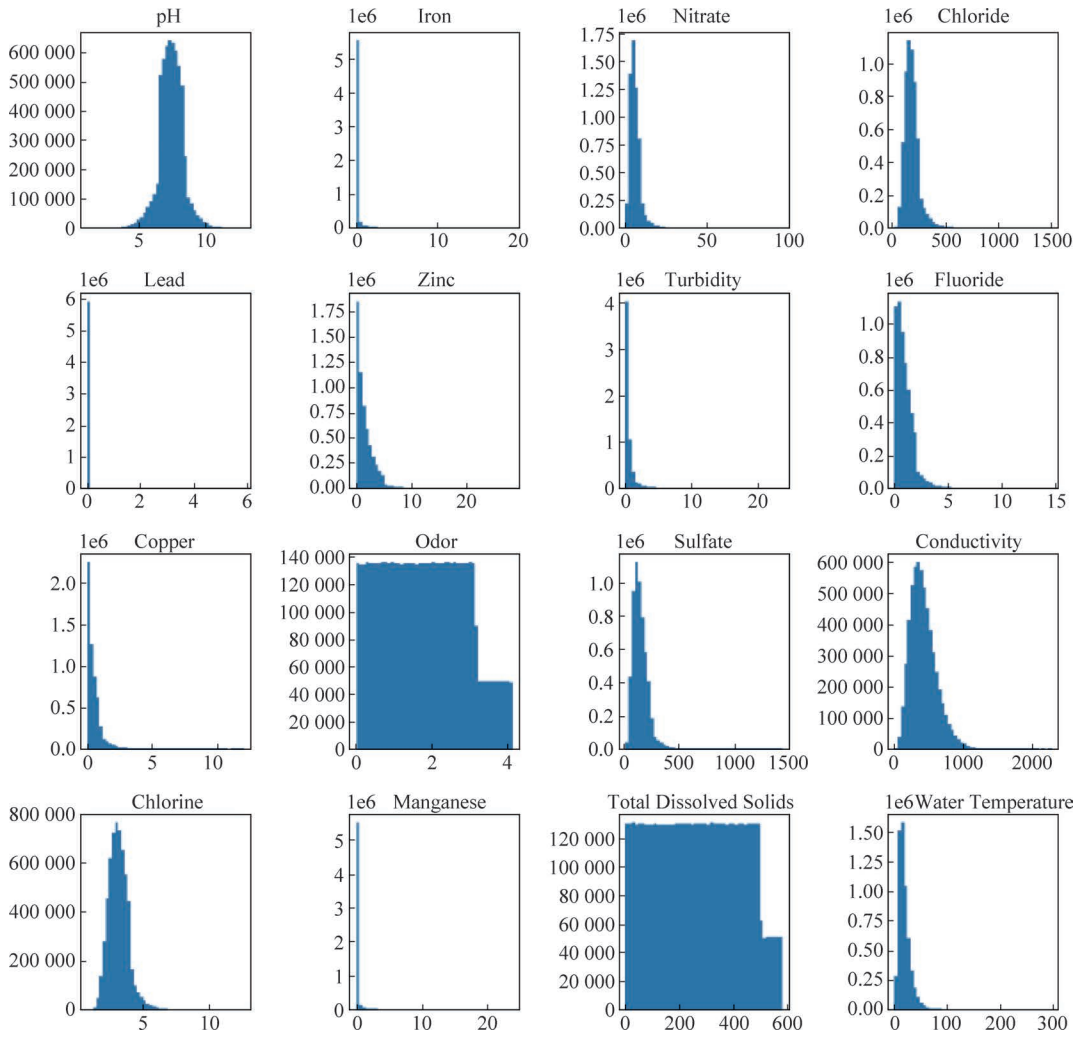

图 5.2　质量分布图

换才能够反映出正态性。例如,部分偏态分布类似于 LogNormal 的分布图像,可以对数据取对数,得到如图 5.3 所示的正态分布。

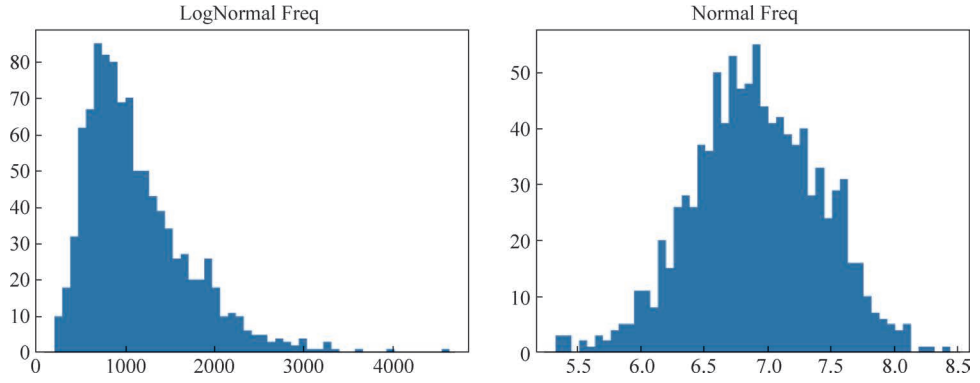

图 5.3　对数正态分布(左)转换成正态分布(右)

5.1.3　异常值检测

异常值指的是分析中录入的错误或者不合常理的数据,对连续值变量可以使用箱线图找出异常值,如图 5.4 所示。

```
def plot_box_woNaN(df,titles):
    plt.figure(figsize=(12,12))
    data=[]
    for title in titles:
        tmp=df[title].dropna()
        data.append(tmp)
    plt.boxplot(data)
    plt.xticks([i+1foriiinrange(len(titles))],titles)
    plt.show()
```

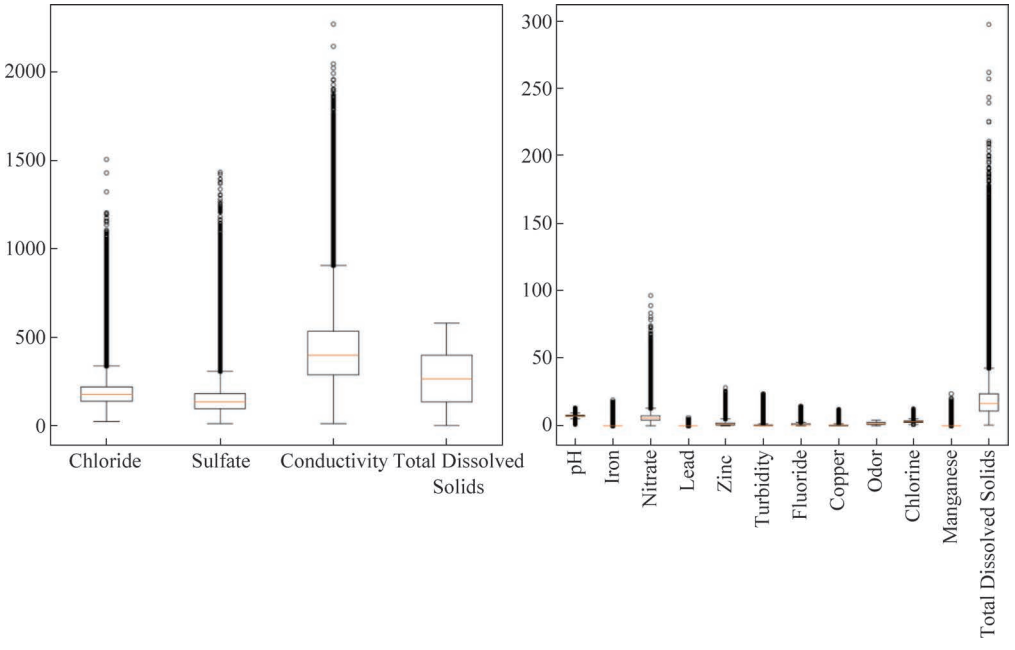

图 5.4　特征箱线图分布

在数据服从正态分布的情况下，异常值通常被定义为一组值外的特定值，这些数值与平均值的偏差超过了 3 倍的标准差。在正态分布假设下，距离平均值 3sigma 外的值可以视为异常值。

5.1.4 相关性检验

数据需要进行相关性检验来判断各个特征之间的独立性，以及是否包含隐藏关系，如图 5.5 所示。

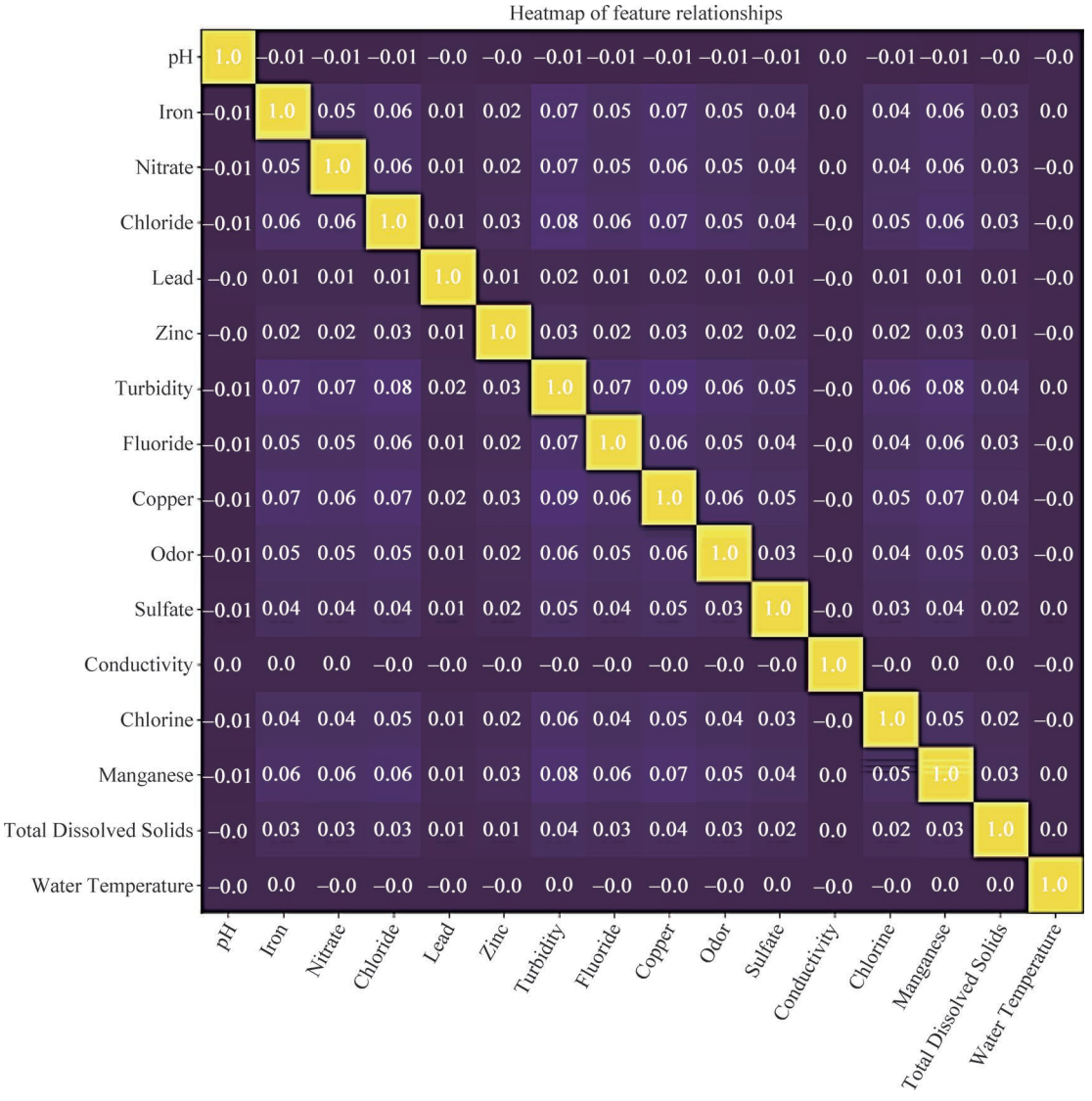

图 5.5 各特征相关系数热力图

当前选取的特征之间的相关性比较弱，独立性较强。

5.1.5 数据离散化

前面讲述的特征矩阵中的属性都是连续属性，而连续属性不适合使用分类算法。因此，对其进行离散化之后，不仅便于分析，并且有助于提高模型的稳定性，降低过拟合的风险。

这里采用的离散方法是 K 均值聚类算法,该算法将参数聚类成 5 个簇:

```
from sklearn.cluster import KMeans
n_clusters = 5
boundary = {}
cluster_table = PrettyTable()
labels = ["parameter","total"]
labels.extend([iforiinrange(n_clusters)])
cluster_table.field_names = labels
fig,axs = plt.subplots(nrows = 4,ncols = 4,figsize = (12,12))
for i I nrange(16):
    row = i//4
    col = i % 4
    ax = axs[row][col]
    param = plot_param[i]
    temp = np.array(df[param]).reshape( - 1,1)
    kmeans = KMeans(n_clusters = 5).fit(temp)
    result = kmeans.predict(temp)
    counts = np.bincount(kmeans.labels_)
    centers = kmeans.cluster_centers_
    boundary[param] = [item[0]foritemincenters]
    boundary[param].sort()
    total = sum(counts)
    row = [param,total]
    for j in range(len(counts)):
        row.append('{}({} % )'.format(round(centers[j][0],3),round(counts[j]/total * 100,
1)))
    cluster_table.add_row(row)
    ax.scatter(result,temp)
    ax.set_title(param)
plt.tight_layout()
plt.show()
```

如图 5.6 所示,当前的标签分布并没有排序,趋向于正态分布的特征,离散化后标签比较均匀。

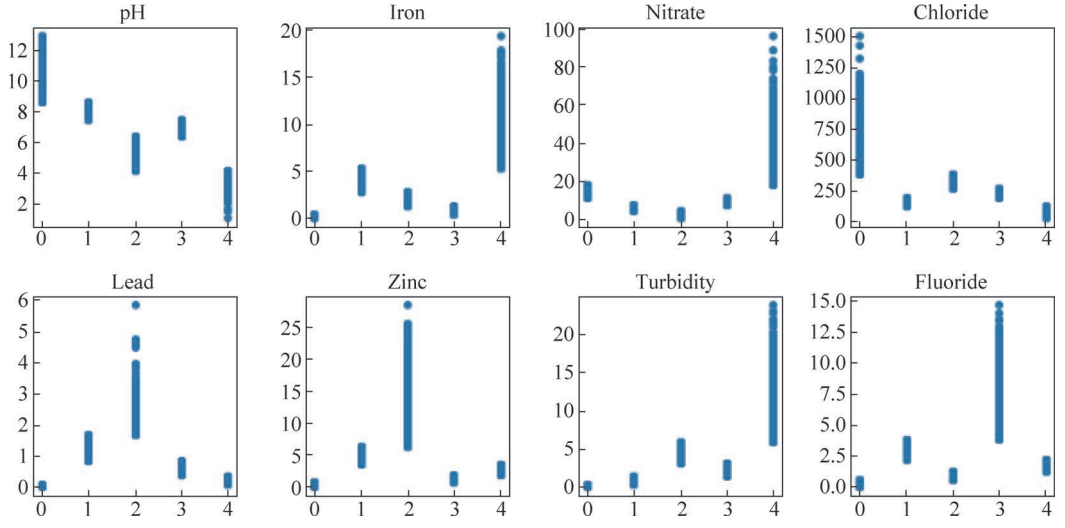

图 5.6　连续数值特征离散化分布图

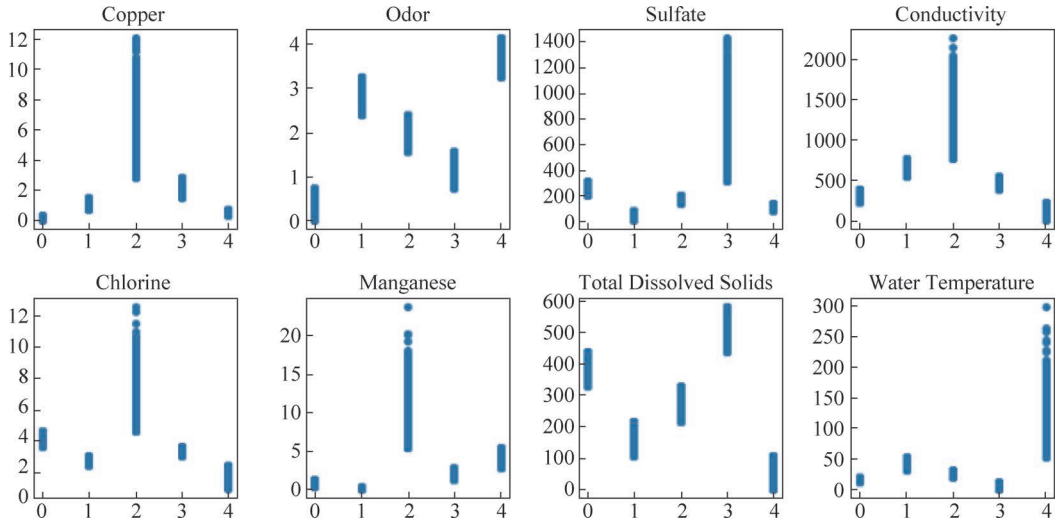

图 5.6 （续）

离散化后数据集中大部分标签所包含的样本点都超过了 99%，也反映出了各个簇的中心位置，如表 5.2 所示。

表 5.2 各特征离散标签占比

parameter	total	0	1	2	3	4
pH	5956842	9.225 (6.7%)	7.962 (41.2%)	5.73 (7.9%)	6.988 (42.1%)	2.536 (2.1%)
Iron	5956842	0.036 (93.3%)	3.663 (0.6%)	1.862 (1.7%)	0.726 (4.3%)	6.987 (0.1%)
Nitrate	5956842	13.536 (4.8%)	5.635 (37.5%)	3.054 (32.0%)	8.539 (24.8%)	22.718 (1.0%)
Chloride	5956842	452.565 (1.8%)	155.521 (39.2%)	306.43 (7.1%)	217.386 (33.1%)	90.894 (18.8%)
Lead	5956842	0.0 (99.7%)	1.155 (0.0%)	2.234 (0.0%)	0.559 (0.1%)	0.199 (0.2%)
Zinc	5956842	0.349 (44.5%)	4.386 (9.1%)	8.267 (1.3%)	1.375 (27.9%)	2.665 (17.3%)
Turbidity	5956842	0.125 (67.4%)	0.784 (24.6%)	4.136 (2.1%)	2.114 (5.4%)	7.686 (0.5%)
Fluoride	5956842	0.277 (41.1%)	2.832 (4.6%)	0.901 (32.2%)	4.809 (1.2%)	1.638 (20.9%)
Copper	5956842	0.118 (47.7%)	0.953 (17.2%)	3.686 (1.2%)	1.956 (4.7%)	0.492 (29.3%)
Odor	5956842	0.329 (23.1%)	2.81 (22.5%)	1.987 (22.8%)	1.155 (22.8%)	3.702 (8.9%)
Sulfate	5956842	232.941 (15.3%)	58.773 (20.0%)	166.716 (27.7%)	395.138 (2.3%)	112.667 (34.8%)
Conductivity	5956842	307.56 (32.6%)	634.733 (17.2%)	895.375 (5.3%)	457.837 (29.2%)	146.322 (15.6%)

续表

parameter	total	0	1	2	3	4
Chlorine	5956842	3.978 (22.3%)	2.769 (29.7%)	5.188 (4.2%)	3.353 (31.2%)	2.084 (12.6%)
Manganese	5956842	0.692 (4.3%)	0.017 (93.3%)	7.018 (0.1%)	1.829 (1.7%)	3.654 (0.6%)
Total Dissolved Solids	5956842	381.784 (20.9%)	159.914 (20.4%)	270.139 (20.9%)	492.53 (17.7%)	52.587 (20.0%)
Water Temperature	5956842	16.268 (36.3%)	39.368 (8.7%)	25.735 (21.5%)	8.301 (31.8%)	64.561 (1.8%)

5.1.6 标签编码

将来源和颜色等原始数据中的输入都转换成整数值。此外,对于之前获得的离散类别的中心值坐标,可以将其转换成分类边界。将其从小到大排列之后,两两取得平均值,就可以获得各个簇的边界点,如图5.7所示。

```
In [21]: #聚类 + 填充

boundary={'pH': [6.133546947745785, 7.106276274862338, 7.819697045992847, 8.776888599296141],
          'Iron': [0.38127865332525807, 1.2945989082716103, 2.7633225845806257, 5.3263707747977875],
          'Nitrate': [4.601350085687351, 7.2605373059972695, 11.18162458047602, 18.301061122216716],
          'Chloride': [140.67118625240082, 195.62100081698213, 267.01045146056487, 382.94199063637154],
          'Lead': [0.09974237089614602, 0.3790696762087246, 0.856850804197288, 1.6946285779001191],
          'Zinc': [0.9180926567692409, 2.096214934198775, 3.6077595637983237, 6.420128535447466],
          'Turbidity': [0.4514498071323091, 1.4312469963531507, 3.0880407803298646, 5.863283158856998],
          'Fluoride': [0.6190715167800545, 1.2897434615119554, 2.2578636191079937, 3.8556104989146074],
          'Copper': [0.31455383842423407, 0.7290830481140268, 1.465557561443609, 2.842737599751058],
          'Odor': [0.815472984223346, 1.623481710948747, 2.4272220621918885, 3.2656449976283204],
          'Sulfate': [101.6456047594126, 152.18193662712707, 210.7892936536984, 325.6481766445204],
          'Conductivity': [280.28995894816927, 420.92788926005073, 575.5125813789565, 787.0230155857241],
          'Chlorine': [2.5722287230379295, 3.144791654915469, 3.7135040830016903, 4.60687900977819],
          'Manganese': [0.35578435357901067, 1.2651839507711178, 2.7540954914139073, 5.364155600581461,
          'Total Dissolved Solids': [105.14409630002616, 212.8755446306955, 323.38927052742724, 435.22447743220505],
          'Water Temperature': [13.547139896752391, 22.15435607563447, 33.809684041556935, 53.56371198377449]
         }
labels={
'Color':['Colorless', 'Faint Yellow', 'Light Yellow', 'Near Colorless','Yellow'],
'Source':['Lake', 'River', 'Ground', 'Spring', 'Stream', 'Aquifer', 'Reservoir', 'Well']
}
```

图5.7 聚类簇边界

```
def discretizing(x,boundary):
    if x:
        for i in range(len(boundary)):
            if x<=boundary[i]:
                return i+1
            return len(boundary)+1
    else:
        return 0

def labelize(x,labels):
    if x:
        ret=1
        for i in labels:
            if x==i:
                return ret
            else:
                ret+=1
    return 0
```

```
def preprocess(df,boundary,labels,droplist):
    df = df.drop(columns = droplist)
    params = df.columns
    ret = {}
    for param in params:
        print(param,'isprocessing')
        if paraminlabels:
            tmp = df[param]
            ret[param] = tmp.apply(lambdax:labelize(x,labels[param])).tolist()
        elif paraminboundary:
            tmp = df[param]
            ret[param] = tmp.apply(lambdax:discretizing(x,boundary[param])).tolist()
        else:
            ret[param] = df[param].tolist()
    return pd.DataFrame(ret)

df = preprocess(rawData,boundary,labels,droplist)
```

使用 preprocess()函数处理，原始的数据离散化如图 5.8 所示。

	pH	Iron	Nitrate	Chloride	Lead	Zinc	Color	Turbidity	Fluoride	Copper
count	5.956842e+06	5.956842e+06	5.956842e+06	5.956842e+06	5.956842e+06	5.956842e+06	5.956842e+06	5.956842e+06	5.956842e+06	5.956842e+06
mean	3.020542e+00	1.126386e+00	2.062556e+00	2.294126e+00	9.983485e-01	2.017384e+00	2.696403e+00	1.474794e+00	2.026779e+00	1.944509e+00
std	1.038978e+00	5.259086e-01	9.980815e-01	1.064442e+00	3.242213e-01	1.138243e+00	1.400062e+00	8.060060e-01	1.081924e+00	1.099401e+00
min	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	0.000000e+00	1.000000e+00	0.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
25%	2.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
50%	3.000000e+00	1.000000e+00	2.000000e+00	2.000000e+00	1.000000e+00	2.000000e+00	3.000000e+00	1.000000e+00	2.000000e+00	2.000000e+00
75%	4.000000e+00	1.000000e+00	3.000000e+00	3.000000e+00	1.000000e+00	3.000000e+00	4.000000e+00	2.000000e+00	3.000000e+00	3.000000e+00
max	5.000000e+00	5.000000e+00	5.000000e+00	5.000000e+00	5.000000e+00	5.000000e+00	5.000000e+00	5.000000e+00	5.000000e+00	5.000000e+00

图 5.8　离散化的特征数据

5.1.7　采样平衡

在机器学习中，要提高模型的准确性，需要将正例与反例的比例保持一个均衡的状态。当训练数据和预测数据在 7∶3 比例的情况下，通过对较少的数据过采样，或者对较多的数据欠采样的方式来进行。

```
size2 = [np.sum(targety == 0),np.sum(targety == 1)]
plt.pie(size2,labels = ['0','1'],autopct = '%1.1f%%')
plt.axis('equal')
plt.show()
```

原始的数据中正例的比值还是相对较少的，因此寻找正例提高召回率显得更加重要，即在训练案例中提高正例的比重。对离散化后的数据进行分割，如果两者比例相差太大，可以使用 down_sample()函数进行处理，对取值多的样本进行欠采样，再对数量少的样品进行过采样。

```
def down_sample(data,ratio = 8.0):
    sampleA = df[df.Target == 1]
```

```
        sampleB = df[df.Target == 0]
        label = ''
        if len(sampleA)>= len(sampleB):
            majority = sampleA
            minority = sampleB
            label = '淡水'
        else:
            majority = sampleB
            minority = sampleA
            label = '非淡水'
            print('currentmajorityis',label,',ratiois',len(majority)/len(minority))
        if len(majority)> len(minority) * ratio:
            count = len(minority) * ratio
            majority_new = pd.resample(majority,replace = False,n_samples = count,random_state =
114514)
            majority = majority_new
        ret = pd.concat([majority,minority])
        return ret
    #数据分割
    from sklearn.model_selection._split import KFold,train_test_split
    df = down_sample(df)#balancethesamplecases
    Y = df['Target']
    X = df.drop(columns = ['Target'])
    A,A2,B,B2 = train_test_split(X,Y,random_state = 1314520,test_size = 0.3)
    #训练数据
    trainx = A.values
    trainy = B.values
    #校验数据
    targetx = A2.values
    targety = B2.values
```

在这里采取对正例过采样的方法进行。这里使用 imbalanced-learn 库中的 RandomOverSampler 来进行过采样处理。

```
from imblearn.over_sampling import RandomOverSampler as ros
ros = ros(random_state = 1314)
X_resample,Y_resample = ros.fit_resample(trainx,trainy)
trainx,trainy = X_resample,Y_resample
```

5.2 模型的训练

淡水的预测识别是一个分类算法问题。常见的分类算法包括决策树、朴素贝叶斯、逻辑回归以及集成学习算法中的随机森林和 XGBoost。这些集成算法可以在多核 CPU 上并行训练。

5.2.1 模型训练与预测

根据前面选择的模型,分别初始化相应的模型,对相同的数据集进行训练,训练好之后对测试数据集进行预测,并记录下预测的结果便于对比:

```
#randomforest
from sklearn.ensemble import Random ForestClassifier
#RandomForest
oldRandomForestClf = RandomForestClassifier(n_estimators = 150, max_depth = 15, min_samples_
split = 7, max_features = 15, n_jobs = -1)
#traindata
oldRandomForestClf.fit(trainx, trainy)
#predictdata
predOldRf = oldRandomForestClf.predict(targetx)
y2_scoreOldRf = oldRandomForestClf.predict_proba(targetx)[:,1]
```

随机森林的初始化参数中部分参数含义如下：

- n_estimators 属于基分类器的数目，集成学习是由多个基分类器分别抽取特征训练，最终综合输出的模型。
- max_depth 用于控制决策树的分叉的深度，防止模型过拟合。
- max_features 用于指定每棵树学习随机抽取的特征数量的上限。如果设置为全特征，随机森林则回归到装袋法集成学习。

以上具体参数通过各个搜索优化而来。

```
import xgboost as xgb
xgb_clf = xgb.XGBClassifier(tree_method = 'gpu_hist', gpu_id = 0)
#traindata
xgb_clf.fit(trainx, trainy)
#predictdata
predxgb = xgb_clf.predict(targetx)
scorexgb = xgb_clf.predict_proba(targetx)[:,1]
#决策树
From sklearn.tree import DecisionTreeClassifier
decisionTreeClf = DecisionTreeClassifier(max_depth = 4)
#traindata
#决策树
From sklearn.tree import DecisionTreeClassifier
decisionTreeClf = DecisionTreeClassifier(max_depth = 4)
#traindata
decisionTreeClf.fit(X_resample, Y_resample)
#predictdata
predTree = decisionTreeClf.predict(targetx)
scoreTree = decisionTreeClf.predict_proba(targetx)[:,1]
from sklearn.naive_bayes import MultinomialNB
#Bayes
nbClf = MultinomialNB(alpha = 0.01)
#traindata
nbClf.fit(X_resample, Y_resample)
#predictdata
predBys = nbClf.predict(targetx)
y2_scoreBys = nbClf.predict_proba(targetx)[:,1]
from sklearn.linear_model import LogisticRegressionCV
lr = LogisticRegressionCV(max_iter = 3000)
#traindata
lr.fit(X_resample, Y_resample)
```

```
#predictdata
predlr = lr.predict(targetx)
y2_scorelr = lr.predict_proba(targetx)[:,1]
```

5.2.2 模型的优化

对于集成学习算法 XGBoost 以及随机森林的参数设置,可以通过格搜索(Grid Search)并行地计算不同的超参数设置对模型的影响,并选择输出最优的模型,提高模型的准确性。

```
from sklearn.metrics impor make_scorer
from sklearn.metrics import accuracy_score,recall_score,precision_score
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import GridSearchCV
param_grid = {
'min_samples_split':range(5,10),#{'min_samples_split':7}
'n_estimators':[100,150,200],#{'n_estimators':150}
'max_depth':[5,10,15],#{'max_depth':15}
'max_features':[5,10,20]#{'max_features':10}
              }
scorers = {
'precision_score':make_scorer(precision_score),
'recall_score':make_scorer(recall_score),
'accuracy_score':make_scorer(accuracy_score),
              }
model = RandomForestClassifier(oob_score = True,max_depth = 10,random_state = 230525)
param_dist = {
'max_depth':range(2,10,1),
'n_estimators':range(60,160,20),
'learning_rate':[0.1,0.01,0.05]
              }
model = GridSearchCV(xgb_clf,param_dist,scoring = 'f1',cv = 5,n_jobs = -1)
```

XGBoost 的优化参数包括决策树的深度、基分类器的数量及学习率。根据格搜索优化的结果,随机森林的优化参数为

```
'min_samples_split':7;
'n_estimators':150;
'max_depth':15;
'max_features':10
```

XGBoost 的优化参数结果为

```
'learning_rate':0.05;
'max_depth':9;
'n_estimators':140;
```

5.3 模型评估

构造对象用于存储测试结果以及各个算法的预测结果,统一计算分数。

```
Class model_evaluator:
    def__init__(self,y):
```

```
            self.target = y
            self.names = []
            self.pred = []
            self.prob = []
            self.trainT = []
            self.predT = []
            self.count = 0
        def clear(self):
            self.names = []
            self.pred = []
            self.prob = []
            self.trainT = []
            self.predT = []
        def add(self,name,pred,prob,trainT = 0.0,predT = 0.0):
            self.names.append(name)
            self.pred.append(pred)
            self.prob.append(prob)
            self.trainT.append(trainT)
            self.predT.append(predT)
            self.count = self.count + 1
        def score(self):
            self.table = PrettyTable()
            self.table.field_names = ["Name","Recall","Precision","ROC",'F1','Traintime','PredictTime']
            recall = recall_score(self.target,[1for_inrange(len(self.target))])
            prec = precision_score(self.target,[1for_inrange(len(self.target))])
            roc = 0.5
            f1 = f1_score(self.target,[1for_inrange(len(self.target))])
            self.table.add_row(['Baseline',recall,prec,roc,f1,0,0])
            for i inrange(self.count):
                recall = recall_score(self.target,self.pred[i])
                prec = precision_score(self.target,self.pred[i])
                roc = roc_auc_score(self.target,self.prob[i])
                f1 = f1_score(self.target,self.pred[i])
                recall = round(recall,4)
                prec = round(prec,4)
                roc = round(roc,4)
                f1 = round(f1,4)
```

如图 5.9 所示，通过对集成学习算法 XGBoost 和随机森林与传统的决策树、朴素贝叶斯和逻辑回归算法比较可以得出结论：随机森林 XGBoost 算法可以获取比较高的准确率、召回率和 F1 值。

Name	Recall	Precision	ROC	F1	Train time	Predict Time
Baseline	1.0	0.3033346296157319	0.5	0.4654746720037131	0	0
DecisionTree	0.6070552998155673	0.7125404801295364	0.7523319374893886	0.655581779465464	13.357882022857666	0.2281808853149414
Nature Bayes	0.6357098409080686	0.44785664108509526	0.7009508187185153	0.5254995956134126	2.7079057693481445	3.2621102333068848
Logic Regression	0.7031158627101108	0.5849008357240864	0.8071067810687241	0.6385834224198392	42.54558038711548	42.869367361068726
Random Forest	0.920210373213777	0.7180908557713263	0.9056120865778573	0.8066827305013773	499.4749312400818	504.7433650493622
XGBoost	0.9196855647688593	0.7175071103214816	0.9026889397813936	0.8061127343429771	96.45919489860535	97.34777665138245

图 5.9　各个算法的评分汇总

各算法的 ROC 曲线如图 5.10 所示。

对于训练好的模型，可以通过如下函数查看模型各个特征的重要性排序，以此判断哪些

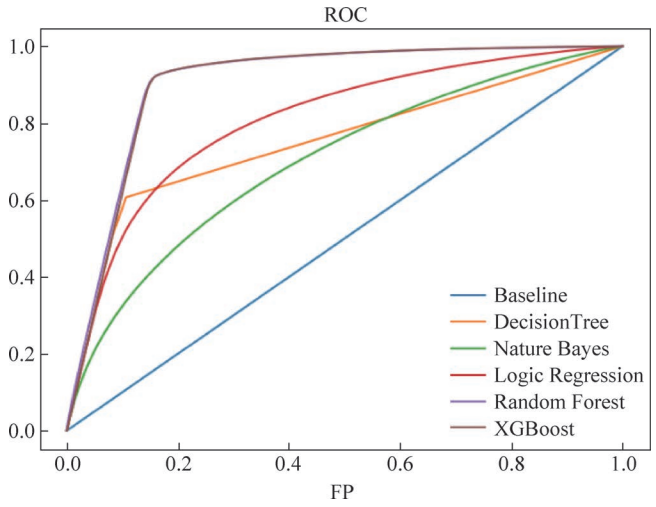

图 5.10　各算法的 ROC 曲线

特征对模型的结果占据主要的影响,如图 5.11 和图 5.12 所示。对于重要性特别弱的特征,可以考虑在模型中剪枝来减少模型复杂度,提高算法速度。

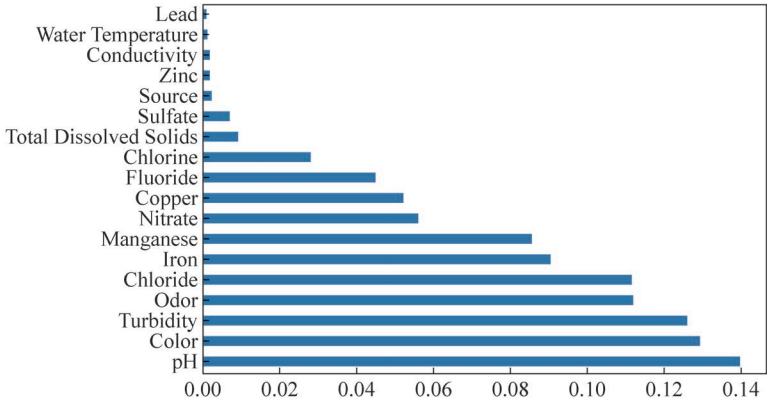

图 5.11　随机森林特征的重要性

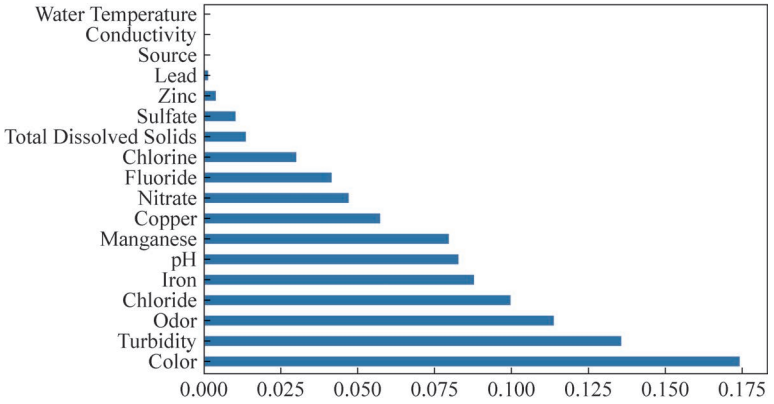

图 5.12　XGBoost 特征的重要性

```
feat_importances = pd.Series(model.feature_importances_, index = X.columns)
feat_importances.nlargest(25).plot(kind = 'barh')
print(X.columns)
```

综合上述两个算法可以得到重要性最高的特征，主要包括水质的颜色、浑浊度、气味、pH 以及锰、铁金属元素。

思　考　题

1. 讨论如何处理不平衡的数据。
2. 讨论缺失数据的处理方法。
3. 如何采用格搜索来优化分类算法的参数？
4. 讨论常见分类算法的优缺点。
5. 如何处理多种分类算法的结果？