成　绩	
评阅人	

复 旦 大 学

研 究 生 课 程 论 文

论文题目：　　预测淡水质量

修读课程：　　机器学习（MSE620023）

选课学期：　　2022-2023 学年第二学期

选课学生：　　韩佳悦（22262010013）

完成日期：　　2023.6.9

预测淡水质量

水是世界上最普遍的物质之一，总体积为 14.1 亿立方公里，其中只有 3%是淡水。淡水的 87%又被封冻在两极及高山的冰层和冰川中，难以利用。便于人类利用的淡水资源只有 21000 立方公里左右。淡水资源几乎触及我们日常生活的方方面面，从饮用、游泳和沐浴到生产食物、电力和我们每天使用的产品。

这些资源在时空上分布不均，加上人类的不合理利用，使世界上许多地区面临着严重的水资源危机。在许多干旱和半干旱地区，淡水成为决定经济发展的重要限制因素，各相关部门之间、地区之间和国家之间争夺淡水资源的情况越来越突出。在水资源比较丰富的地区，不同功能用途之间的矛盾和冲突也越来越显著。获得安全卫生的供水不仅对人类生活至关重要，而且对正在遭受干旱、污染和气温升高影响的周边生态系统的生存也至关重要。

1. 数据预处理

采集完的数据不能直接使用，数据中存在无法直接处理的字符格式数据，缺失空白数据和超出范围的可能错误数据需要处理，不适合直接开展机器学习工作，因此需要进行数据预处理。

1.1 数据整体情况的初步分析：

调用 info 函数得到特征，特征类型和数据大小等信息，如下 Figure 1:

```
1.  datas = df.drop_duplicates()
2.  datas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5956842 entries, 0 to 5956841
Data columns (total 24 columns):
 #   Column                 Dtype
---  ------                 -----
 0   Index                  int64
 1   pH                     float64
 2   Iron                   float64
 3   Nitrate                float64
 4   Chloride               float64
 5   Lead                   float64
 6   Zinc                   float64
 7   Color                  object
 8   Turbidity              float64
 9   Fluoride               float64
 10  Copper                 float64
 11  Odor                   float64
 12  Sulfate                float64
 13  Conductivity           float64
 14  Chlorine               float64
 15  Manganese              float64
 16  Total Dissolved Solids float64
 17  Source                 object
 18  Water Temperature      float64
 19  Air Temperature        float64
 20  Month                  object
 21  Day                    float64
 22  Time of Day            float64
 23  Target                 int64
dtypes: float64(19), int64(2), object(3)
memory usage: 1.1+ GB
```

(Figure 2)

将这些数据内容归纳整理进表格进行进一步分析，如下表：

#	Column	Meaning	Dtype
0	Index	索引	int64
1	pH	酸碱度	float64
2	Iron	铁	float64
3	Nitrate	硝酸盐	float64
4	Chloride	氯化物	float64
5	Lead	铅	float64
6	Zinc	锌	float64
7	Color	颜色	object
8	Turbidity	浑浊度	float64
9	Fluoride	氟化物	float64
10	Copper	铜	float64
11	Odor	气味	float64
12	Sulfate	硫酸盐	float64
13	Conductivity	传导率	float64
14	Chlorine	氯	float64
15	Manganese	锰	float64
16	Total Dissolved Solids	不可溶物总含量	float64
17	Source	水源	object
18	Water Temperature	水温	float64
19	Air Temperature	空气温度	float64
20	Month	月份	object
21	Day	日期	float64
22	Time of Day	时间	float64
23	Target	结果	int64

- dtypes: float64(19), int64(2), object(3)
- memory usage: 1.1+ GB

(Table 1)

我们可以看到，在数据集中，除去 index 项，一共有 21 项特征，其中除了 3 项为 object 类型的离散数据，其余为 float64 的连续数据，结果通过 int 型的 Target 表现，target 取值为 0 或 1. 因此。我们可以将“判断淡水是否可用”的这个问题转化为对于 Target 值为”0” or ”1”的二元分类问题，并处理筛选 21 项特征对其进行建模。

“Index”项与建模及预测无关，仅作数据的标号，我们把它设为索引，移出后续对特征的考虑和分析范围。

```
# 将 Index 设置成索引列
datas.set_index("Index", inplace=True)
```

我们进一步对数据中 Target 值分别为 0/1 的数量做一个初步的判断，从结果中我们看到，结果 Target 分别为 0/1 的占比约为 7:3，比例较为合理。

```
print(datas["Target"].value_counts())
print("target 为 0 的占比:
", datas["Target"].value_counts()[0] / (datas["Target"].value_counts(
)[0] + datas["Target"].value_counts()[1]))
```

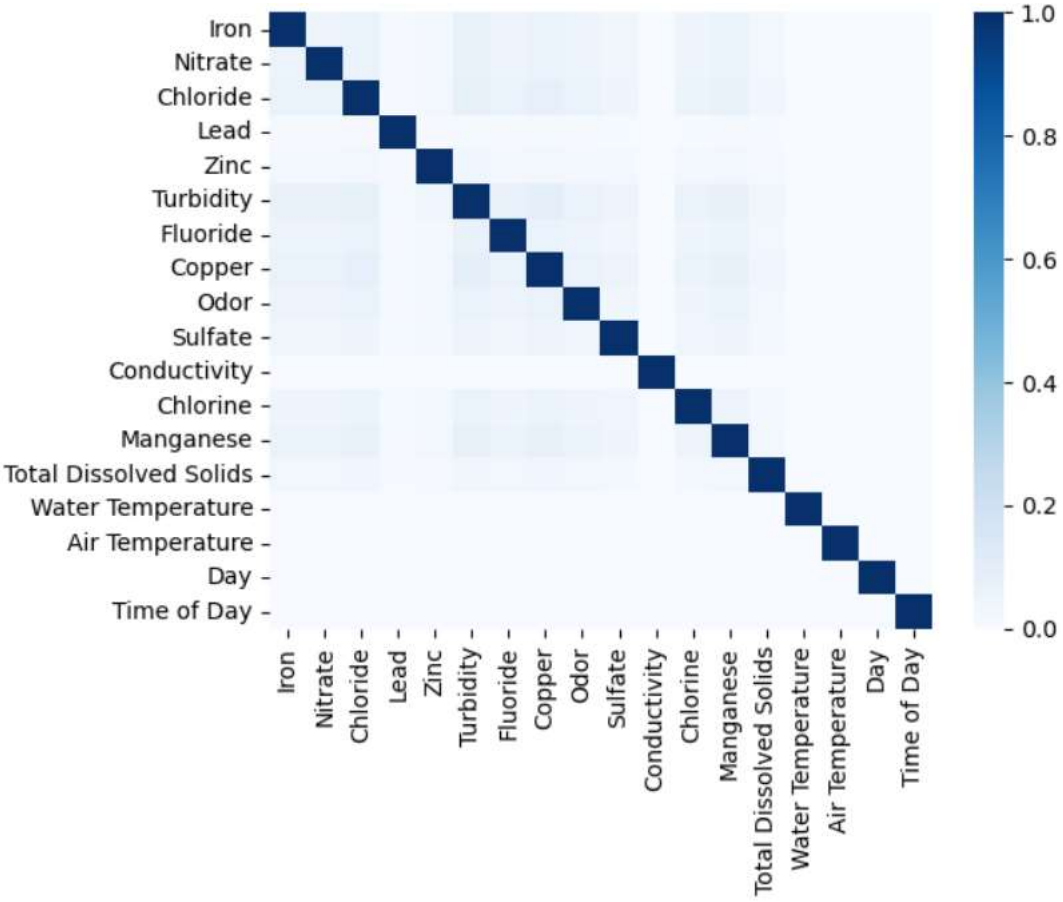
```
print("target 为 0 的占比:
", datas["Target"].value_counts()[1] / (datas["Target"].value_counts(
)[0] + datas["Target"].value_counts()[1]))
```

```
0    4151590
1    1805252
Name: Target, dtype: int64
target为0的占比： 0.6969447905450573
target为0的占比： 0.30305520945494274
```

1.2 特征间相关性计算，尝试合并特征：

对于非 object 类型的数据，我们计算其相互之间的相关性，尝试将关联非常强的数据进行合并，以减少特征数，进而减小模型复杂度，降低过拟合风险。

```
# 数值变量
num_variables = [v for v in datas.columns.tolist() if datas[v].dtype
!= "object"][1:-1]
print(num_variables)
df_num_value = datas[num_variables]
# 计算相关系数
print(df_num_value.corr())
sns.heatmap(df_num_value.corr(), cmap='Blues')
```



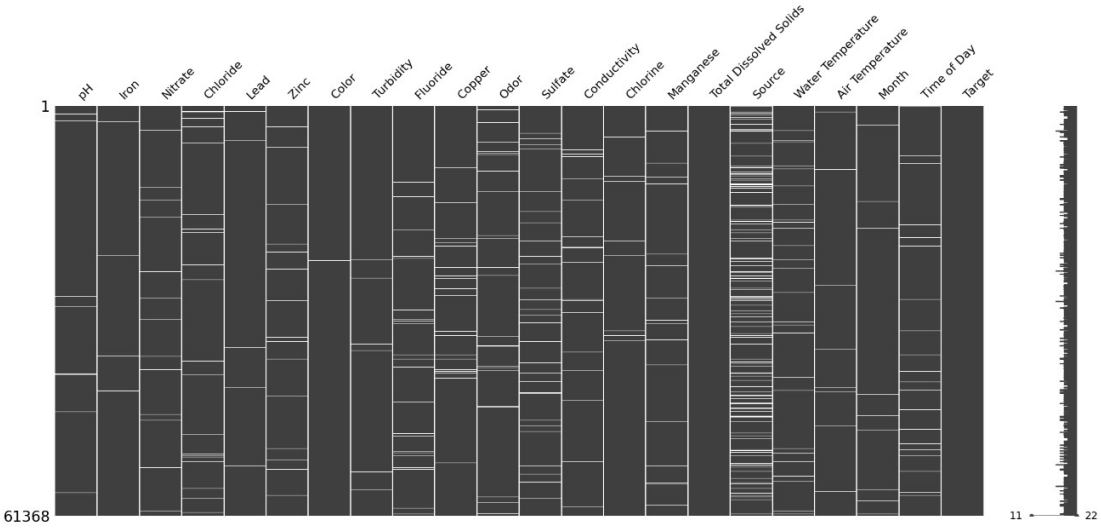
根据显示结果，我们可以看出，特征间除了和自身有强关联外，并不存在其他的数据间强相关的情况，因此此处不做额外的数据合并。

1.3 缺失数据处理

空白缺失的数据会对数据预处理中后续的特征筛选，错误数据评估以及数据平衡等操作都会带来问题和错误，同时也会对整体的模型训练带来非常坏的影响，所以需要

在进行处理之前先对丢失数据做补齐。

使用 MissingNo 库的可视化工具对处理前后的数据进行显示，，可以快速直接的了解到数据处理前后的数据完整性：



可以看到“Source”，“Copper”，“Sulfate”有明显的数据缺失，其余特征项也都有不同程度的空白数据。

对于 object 类型的变量缺失值可以使用众数填充：

```
# 字符变量众数填充
str_variables = [v for v in datas.columns.tolist() if datas[v].dtype == "object"]
datas[str_variables] = datas[str_variables].fillna(datas[str_variables].mode())
```

对于数值型变量缺失值使用 KNN 进行插值补齐数据，计算找到临近的 3 条记录，通过投票选出其中最多的类别用以填充。

```
num_variables = [v for v in datas.columns.tolist() if datas[v].dtype != "object"]
# KNN 填充
fill_knn = KNN(k=3).fit_transform(datas[num_variables])
datas[num_variables] = pd.DataFrame(fill_knn)
msno.matrix(datas)
```

1.4 数据类型转换：

经过上一步对于整体数据情况的初步分析，我们发现有 3 项 object 类型的离散数据，分别为“Color”，“Source”，“Month”。这样的数据无法进行后续的预处理操作，所以需要先对这部分数据做转换。

我们首先提取出这 3 项特征具体所含类别内容分别有哪些，如下：

```
print(datas["Color"].value_counts())
print(datas["Source"].value_counts())
print(datas["Month"].value_counts())
```

```
March        498700
July         498132
May          498043
January      497875
December     497349
August       497072
October      496061
April        482261
June         482016
September    481456
November     481020
February     451189
Name: Month, dtype: int64
```

```
Stream       734502
Ground       734389
Well         734315
Aquifer      733778
Reservoir    733298
River        732980
Spring       732700
Lake         732618
Name: Source, dtype: int64
```

```
Colorless         1787911
Near Colorless    1786234
Faint Yellow      1079772
Light Yellow       758138
Yellow             539048
Name: Color, dtype: int64
```

其中“color”项有大小意义，水的颜色从浅到深，从清澈到浑浊分别为：

“Colorless < Near Colorless < Faint Yellow < Light Yellow < Yellow”

因此我们直接将这 5 中状态与 0~4 的 int 型数值进行对应：

```
1.  datas['Color'] = datas['Color'].map({"Colorless":0, "Near Colorless":
    1, "Faint Yellow":2, "Light Yellow":3,"Yellow":4})
```

Colorless	0
Near Colorless	1
Faint Yellow	2
Light Yellow	3
Yellow	4

“Source”项大小无关，不同水源之间没有明显的等级与优劣关系，因此我们采用独热编码(one-hot)的方法对数据进行转换。

```
1.  Source_1hot = pd.Series(datas['Source'])
2.  Source_1hot = pd.get_dummies(Source_1hot )
3.  print(Source_1hot)
```

```
         Aquifer  Ground  Lake  Reservoir  River  Spring  Stream  Well
0              0       0     0          0      0       0       0     0
1              0       0     1          0      0       0       0     0
2              0       0     0          0      1       0       0     0
3              0       1     0          0      0       0       0     0
4              0       0     0          0      0       1       0     0
...          ...     ...   ...        ...    ...     ...     ...   ...
5956837        0       0     0          0      0       0       0     1
5956838        0       1     0          0      0       0       0     0
5956839        0       0     0          0      0       0       0     0
5956840        0       0     0          0      0       0       0     0
5956841        1       0     0          0      0       0       0     0

[5956842 rows x 8 columns]
```

对于“Month”项，虽然可以与数值直接相对应，但是此项特征对于预测建模而言并没有明确的等级大小相关性，因此这里也采用独热编码的方式对其进行转化。

```
1.  Month_1hot = pd.get_dummies(Month_1hot )
2.  print(Month_1hot)
```

```
         April  August  December  February  January  July  June  March  May  \
0            0       0         0         0        1     0     0      0    0
1            0       0         0         0        0     0     0      0    0
2            0       0         0         0        1     0     0      0    0
3            1       0         0         0        0     0     0      0    0
4            0       0         0         0        0     0     1      0    0
...        ...     ...       ...       ...      ...   ...   ...    ...  ...
5956837      0       0         0         1        0     0     0      0    0
5956838      0       0         0         0        0     0     0      0    0
5956839      0       0         0         0        1     0     0      0    0
5956840      0       0         0         0        0     0     0      1    0
5956841      0       0         0         0        0     0     1      0    0

         November  October  September
0               0        0          0
1               1        0          0
2               0        0          0
3               0        0          0
4               0        0          0
...           ...      ...        ...
5956837         0        0          0
5956838         0        0          1
5956839         0        0          0
5956840         0        0          0
5956841         0        0          0

[5956842 rows x 12 columns]
```

将处理后的数据与原始数据集进行合并处理，得到经过数据转化的新数据集，重新命名为“df_transferred”。

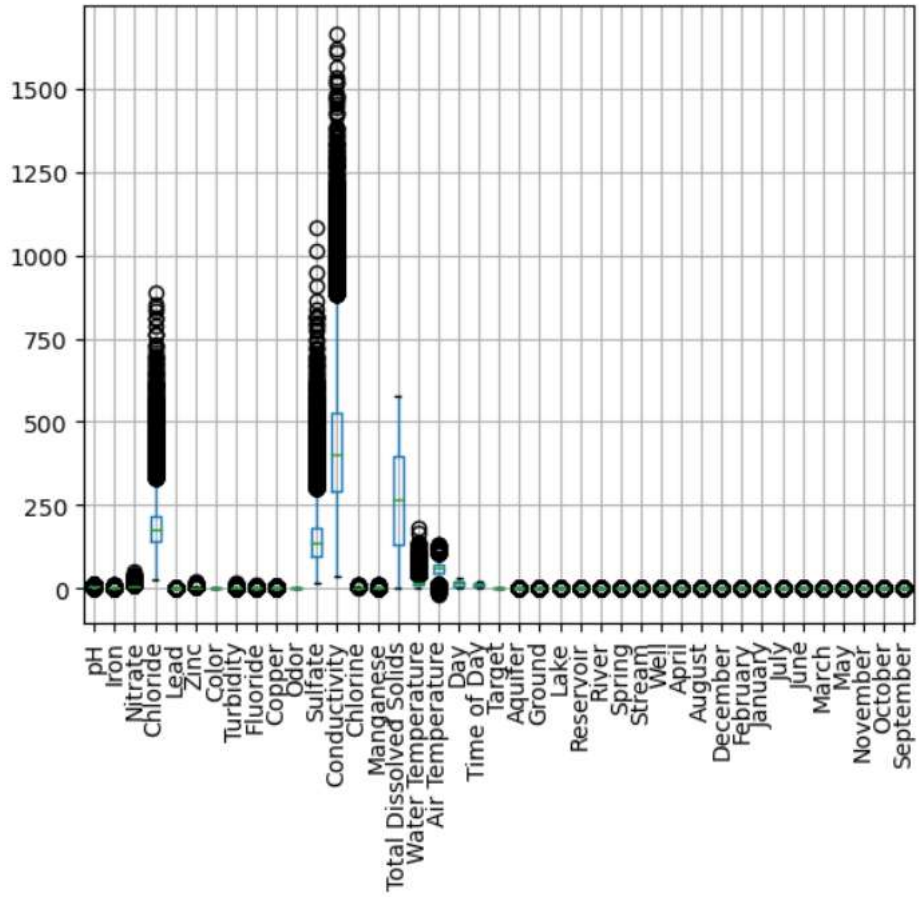
```
Source_1hot = pd.Series(datas['Source'])
dummies_Source = pd.get_dummies(Source_1hot)
Month_1hot = pd.Series(datas['Month'])
dummies_Month = pd.get_dummies(Month_1hot)

df_transferred = datas[["pH", "Iron","Nitrate","Chloride","Lead","Zinc","Color","Turbidity","Fluoride","Copper","Odor","Sulfate","Conductivity","Chlorine","Manganese","Total Dissolved Solids","Water Temperature",
                        "Air Temperature","Day","Time of Day","Target"]].join(dummies_Source).join(dummies_Month)
```

	pH	Iron	Nitrate	Chloride	Lead	Zinc	Color	Turbidity	Fluoride	Copper	...	December	February	January	July	June	March
0	8.332988	0.000083	8.605777	122.799772	3.713298e-52	3.434827	0.0	0.022683	0.607283	0.144599	...	0	0	1	0	0	0
1	6.917863	0.000081	3.734167	227.029851	7.849262e-94	1.245317	2.0	0.019007	0.622874	0.437835	...	0	0	0	0	0	0
2	5.443762	0.020106	3.816994	230.995630	5.286616e-76	0.528280	3.0	0.319956	0.423423	0.431588	...	0	0	1	0	0	0
3	7.955339	0.143988	8.224944	178.129940	3.997118e-176	4.027879	1.0	0.166319	0.208454	0.239451	...	0	0	0	0	0	0
4	8.091909	0.002167	9.925788	186.540872	4.171069e-132	3.807511	3.0	0.004867	0.222912	0.616574	...	0	0	0	0	1	0

1.5 错误数据处理

这里的错误数据指明显超过平均值，和数据集中其他数据相比，有及其明显差异的数据，通过箱图，我们对数据集中的数值数据分布情况做一个判断：

```
# 数值变量
num_variables = [v for v in df_transferred.columns.tolist() if df_transferred[v].dtype != "object"]
df_transferred[num_variables].boxplot()
plt.xticks(rotation=90)
plt.show()
```



发现["Chloride", "Sulfate", "Conductivity"]项有明显的偏移，对其进行调整：

```
for item in ["Chloride", "Sulfate", "Conductivity"';[[[[[[[[[[]]]]]]]]]]]']:
#     for item in [4, 12, 13]:
    q1 = df_transferred[item].quantile(0.25)
    q3 = df_transferred[item].quantile(0.75)
    iqr = q3 - q1
    df_transferred = df_transferred.query(f"(@q1 - 1.5 * @iqr) <= {item} <= (@q3 + 1.5 * @iqr)")
df_transferred.plot.box()
plt.xticks(rotation=90)
plt.show()
```

1.6 数据与结果的相关性判断及特征选取

初始数据集 feature 有 24 项，其中各项特征对于结果的影响程度不同，为了避免对结果影响不大的特征“干扰”模型训练，使得整个模型趋于复杂，我们先对特征本身做分析和筛选。

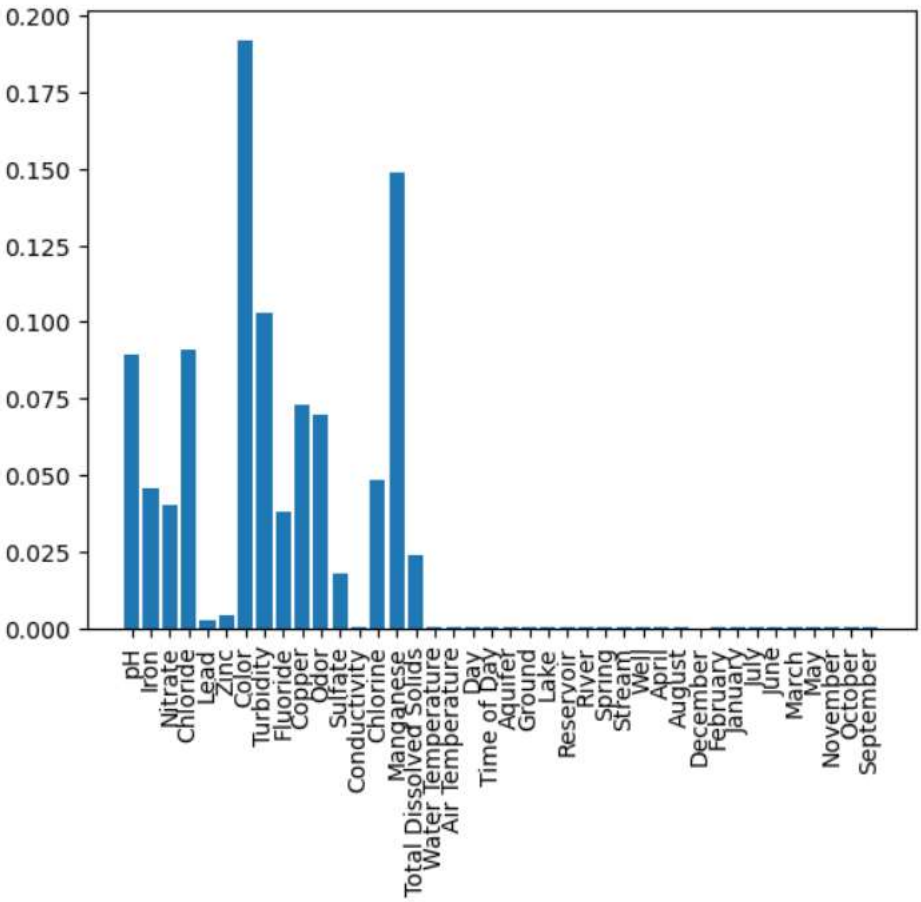
这里我们分别采用基于 XGBoost 的稳定性选择方法实现对特征的筛选和随机森林的方法，生成各项特征对于预测结果影响程度的柱状图。

首先做准备工作，将特征和输出结果拆分开

```
features = ["pH","Iron","Nitrate","Chloride","Lead","Zinc","Color","Turbidity","Fluoride","Copper",
            "Odor","Sulfate","Conductivity","Chlorine","Manganese","Total Dissolved Solids","Water Temperature","Air Temperature","Day","Time of Day","Aquifer","Ground","Lake","Reservoir",
            "River","Spring","Stream","Well","April","August","December","February","January","July",
            "June","March","May","November","October","September"]
A1 = df_transferred[features]
B1 = df_transferred[["Target"]]
x1 = A1.values
y1 = B1.values
print(x1)
print(y1)
```

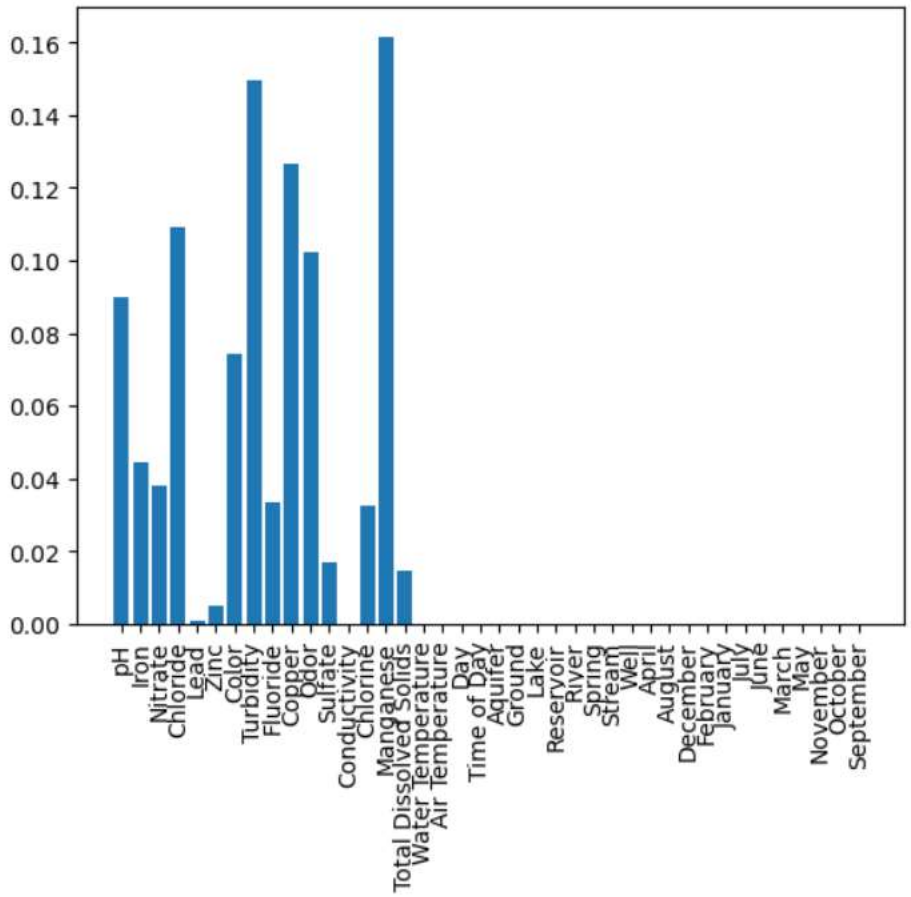
1.6.1 基于 XGBoost:

```
 y1 = LabelEncoder().fit_transform(y1.ravel())
x_train, x_test, y_train, y_test = train_test_split(x1, y1, test_size=0.3, random_state=1)
xgboost = XGBClassifier()
xgboost.fit(x_train, y_train)
# print(list(xgboost.feature_importances_))

plt.bar(features, list(xgboost.feature_importances_))
plt.xticks(rotation=90)
plt.show()
```



1.6.2 基于 Random Forest 随机森林的方法:

```
1.  from sklearn.ensemble import RandomForestClassifier
2.  rf = RandomForestClassifier(n_estimators=100, criterion='gini', max_d
    epth=3)
3.  rf.fit(x_train, y_train)
4.  # print(f"变量重要性{rf.feature_importances_}")
5.
6.  plt.bar(features, list(rf.feature_importances_))
7.  plt.xticks(rotation=90)
8.  plt.show()
```



根据此步判断的结果，我们可以发现，预测的结果与以下特征的相关性较大:

["pH","Iron","Nitrate","Chloride","Color","Turbidity","Fluoride","Copper","Odor","Sulfate",
" Chlorine","Manganese","Total Dissolved Solids"]

在后续处理中，我们可以考虑基于这部分特征值进行建模和分析，以尽可能减小模型复杂度，减少过拟合的风险。

1.7 数据平衡

当前的数据集中，淡水是否可用的结果值仍然分布非常不均，若直接进行训练集和测试集的划分，会造成训练集中的数据严重不平衡，进而导致模型训练结果偏差，因此需要进行数据平衡，尤其是对于训练集的数据平衡。

在处理前，我们将训练集和测试集的数据分布情况先做一个输出，通过“Target”为 0 的数据项在总体中的占比来提现数据分布的平衡性，根据输出结果可以发现，在进行数据平衡前，训练集和数据集中的数据分布及其不均，几乎只有”Target”值为 1 的数据项，显然这样的数据无法训练模型。

这里我们使用基于调整算法分 SMOTE 算法进行数据平衡处理，Target 为 0 的数据占比提升至 0.4，”Target”分别为 0,1 的数据的分布比例提升为大约 4:6，达到一个合理平衡的数据分布状态。

```
print(" #### BEFORE: 训练集数据平衡性 ####")
print(y_train_a.tolist().count(0.0) / (y_train_a.shape[0]))
print(y_test_a.tolist().count(0.0)/ (y_test_a.shape[0]))

print(" #### AFTER: 训练集数据平衡性 ####")
```

```
6.  X_train_a, y_train_a = SMOTEENN().fit_resample(X_train_a, y_train_a)
```

```
 #### BEFORE: 训练集数据平衡性 ####
0.0
0.0
 #### AFTER: 训练集数据平衡性 ####
0.374443903231044
0
```

2. 建模和训练

2.1 模型的选择：

在完成数据预处理后，开始对淡水是否可用的规则进行分析。如前文 1.1 中所述，我们可以将“判断淡水是否可用”的这个问题转化为对于 Target 值为”0” or ”1”的二元分类问题，于是考虑使用随机森林的方法对淡水是否可用的预测进行分析。 这里我们分别尝试使用完整的特征和仅相关性强的特征的数据集进行建模和训练，并使用准确率指标对模型性能进行初步的评估。

2.2 模型的建立与训练：

2.2.1 使用完整特征的数据集进行建模和训练：

使用传统的随机森林方法，多次调整最大深度参数，得到在深度为 20 左右时，准确率基本维持在较高的状态，切再增加深度对准确率提升效果有限：

```
1.  # print(X_train, X_test, y_train, y_test)
2.  # 随机森林分类器
3.  rf = RandomForestClassifier(n_estimators=100, criterion='gini', max_d
    epth=20)
4.  rf.fit(X_train_a, y_train_a)
5.  # 测试集预测
6.  y_hat = rf.predict(X_test_a)
7.  # 模型评估
8.  print("Accuracy:",metrics.accuracy_score(y_test_a,y_hat))
9.  print(metrics.classification_report(y_test_a,y_hat))
```

```
Accuracy: 0.8787666825214739
```

2.2.2 仅使用相关性强的特征的数据集进行建模和训练：

与 2.2.1 相似，得到模型和准确率如下：

```
1.  # 随机森林分类器
2.  rf = RandomForestClassifier(n_estimators=100, criterion='gini', max_d
    epth=20)
3.  rf.fit(X_train_s, y_train_s)
4.  # 测试集预测
5.  y_hat = rf.predict(X_test_s)
6.  # 模型评估
7.  print("Accuracy:",metrics.accuracy_score(y_test_s,y_hat))
```

```
Accuracy: 0.8798298872443412
```

2.2.3 传统随机森林方法小结：

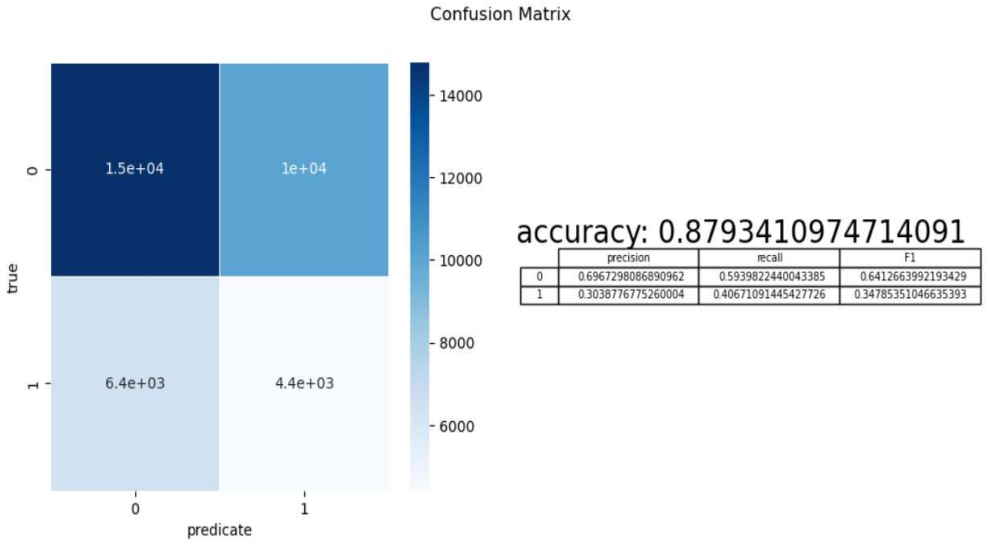
综上，传统的随机森林方法有良好的适应性，使用传统随机森林的方法对数据进行建模和训练，预测结果的准确率可以达到越 87.9%，性能较好。

3. 模型评估

前文在建模和训练时仅使用准确率作为判断，较为单一和片面，我们在本文的最后一段模型评估中，从 f1 值和 ROC 曲线的方面再次对整体模型做一个评估。

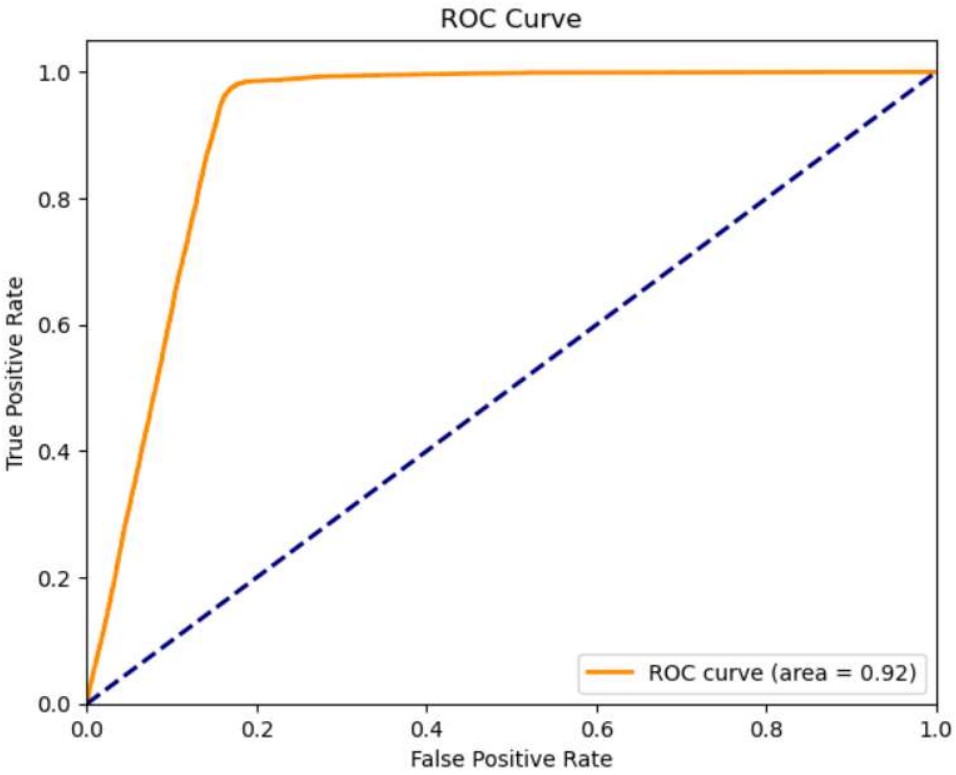
3.1 混淆矩阵和模型评估参数：

```
    from sklearn.metrics import confusion_matrix
cm1 = confusion_matrix(y_test, y_hat)
print(cm1.shape) # (2, 2)
print(cm1)

fig = plt.figure(figsize=(12,5))
fig.suptitle(r'Confusion Matrix')
#画热力图
ax1 = fig.add_subplot(1,2,1)
sns.heatmap(cm1,annot=True,cmap='Blues',linewidths=0.5,ax=ax1)
ax1.set_xlabel('predicate')
ax1.set_ylabel('true')

(tn,fp,fn,tp) = cm1.ravel()
print((tn,fp,fn,tp))

acc = metrics.accuracy_score(y_test_s,y_hat_1)
pre0 = tn/(fn+tn)
pre1 =tp/(tp+fp)
recall0 = tn/(fp+tn)
recall1 = tp/(tp+fn)
f10 = 2*tn/(2*tn+fp+fn)
f11 = 2*tp/(2*tp+fp+fn)

col_labels = ['precision','recall','F1']
row_labels = [' '*4+'0'+' '*4,' '*4+'1'+' '*4]
table_vals = [[pre0,recall0,f10],[pre1,recall1,f11]]
ax2 = fig.add_subplot(1,2,2)
ax2.table(cellText=table_vals,rowLabels=row_labels,
          colLabels=col_labels,loc='center')
ax2.text(0.35,0.6,
         r'''''
        accuracy: {acc}
          '''.format(acc=acc),horizontalalignment='center',verticalal
ignment='center',fontsize=20, color='black',transform=ax2.transAxes)

ax2.axis('off')
ax2.set_xticks([])
ax2.set_yticks([])
plt.show()
```



3.1.2 ROC 曲线和 AUC

```
    from sklearn.metrics import roc_curve
from sklearn import metrics

y_score = rf.fit(X_train_a, y_train_a).predict_proba(X_test_a)  # 随机森林
fpr, tpr, thresholds = roc_curve(y_test_a, y_score[:, 1])
roc_auc = metrics.auc(fpr, tpr)

plt.subplots(figsize=(7, 5.5))
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```



通过图形，我们可以看到 ROC 曲线光滑，该模型的过拟合现象轻。 ROC 曲线下方面积(AUC)值较大，到达了 0.92，说明模型的预测准确性高。