KOREA UNIVERSITY

Predicting water potability using machine learning

Midterm Project Presentation

WOO-CHAN KIM

KOREA UNIVERSITY

I. Project title

Predicting Water Potability Using Machine Learning

II. Project introduction

1) Objective

: Developing a machine learning model to predict water potability based on various water quality indicators.

2) Motivation

(1) I chose this project because it is meaningful work that would make a social contribution like public health improvement, environmental protection and sustainability .

(2) It has correlations with my first project of movie review classifier. So, I could get a deeper knowledge of classification.

III. Analysis of original code

1) Dataset

- Train: 1,608
- Validation: 201
- Test: 202

2) Machine Learning Model

: Xgboost

3) Performance

- Accuracy : 0.6584
- F1-score(0): 0.74
- F1-score(1): 0.51

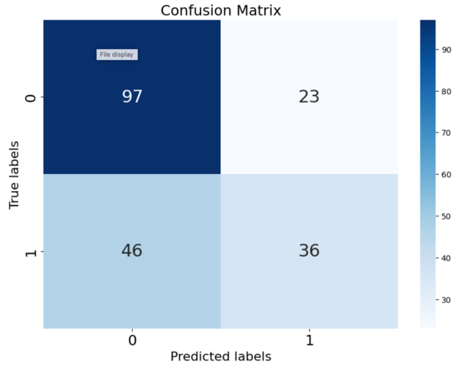
```python
y_pred = best_xgb.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.4f}")
```

```
Test Accuracy: 0.6584
```

```python
# Generate and print the classification report
report = classification_report(y_test, y_pred)
print(f"Classification Report:\n{report}")
```

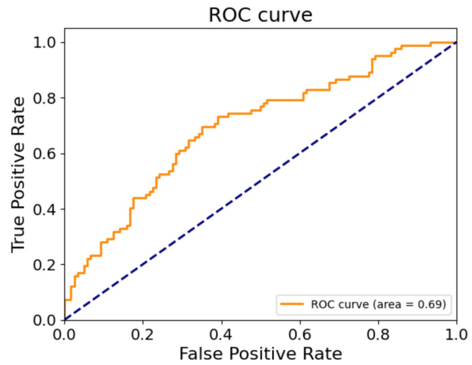
```
Classification Report:
              precision    recall  f1-score   support

           0       0.68      0.81      0.74       120
           1       0.61      0.44      0.51        82

    accuracy                           0.66       202
   macro avg       0.64      0.62      0.62       202
weighted avg       0.65      0.66      0.65       202
```

Origin Code



Confusion Matrix

ROC AUC Score: 0.6944



ROC curve

IV. Updating Preprocessing Process

1) Dealing with missing values

: Replace missing values with a representative value rather than deleting them.

III. Dataset Description

Water Potability Dataset

In this project, a total of 3,276 data samples are preprocessed and 2011 data are divided into training sets, validation sets, and test sets for model development.

Training Set:

Composed of 1,608 samples. This dataset is used to train the machine learning model. The model learns the relationship between various water quality indicators and potability during this stage.

Validation Set:

Composed of 201 samples. This set is used to evaluate model performance during training, allowing for tuning of hyperparameters and preventing overfitting. It helps monitor the model's generalization performance and adjust the learning process accordingly.

Test Set:

Composed of 202 samples. After the model is fully trained, this set is used to assess the final performance of the model. It provides an evaluation of how well the model predicts water potability in real-world scenarios.

- **Origin Dataset:** 3,276
- **Sum of train/validation/test sets:** 2,011

- **The number of missing data:** 1,265

```
# Check the missing value percentage
missing_values = df.isnull().sum()
missing_percentage = (df.isnull().sum() / len(df)) * 100

# Generate Missing value info table
missing_summary = pd.DataFrame({
    'Missing Values': missing_values,
    'Percentage (%)': missing_percentage
})

print(missing_summary)
```

```
                 Missing Values  Percentage (%)
ph                          491       14.987790
Hardness                      0        0.000000
Solids                        0        0.000000
Chloramines                   0        0.000000
Sulfate                     781       23.840049
Conductivity                  0        0.000000
Organic_carbon                0        0.000000
Trihalomethanes             162        4.945055
Turbidity                     0        0.000000
Potability                    0        0.000000
```

Detecting missing values

```
[7] # Replace missing values by Mean
    df['ph'] = df['ph'].fillna(df['ph'].mean())

[8] # Replace missing values by Mean
    df['Sulfate'] = df['Sulfate'].fillna(df['Sulfate'].mean())

[9] # Replace missing values by Mean
    df['Trihalomethanes'] = df['Trihalomethanes'].fillna(df['Trihalomethanes'].mean())
```

Replace missing value
to mean value

IV. Updating Preprocessing Process

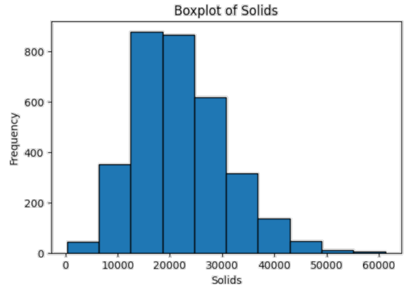
2) Dealing with outlier values

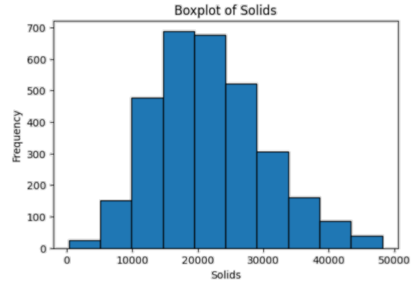
```
[ ] from scipy import stats

    outlier_indices = set()

    # Z-Score
    for column in df.select_dtypes(include='number').columns:
        z_scores = np.abs(stats.zscore(df[column]))
        outliers = df[z_scores > 3]
        outlier_indices.update(outliers.index)
        print(outliers)
```

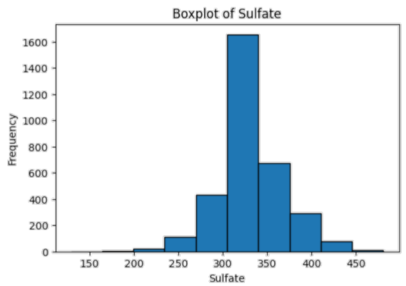
```
print("Outlier Index:", outlier_indices)
print(f"The number of Outliers: {len(outlier_indices)}")
```

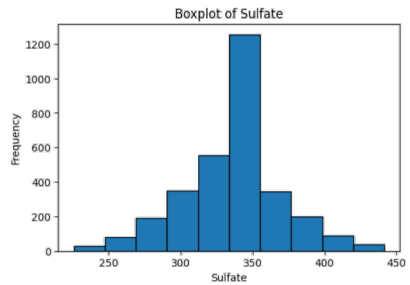
```
df = df.drop(index=outlier_indices)
```

Code









Before deleting outliers

After deleting outliers

IV. Updating Preprocessing Process

3) Data Normalization & Splitting the dataset

```
[47] from sklearn.preprocessing import MinMaxScaler

     scaler = MinMaxScaler()
     normalized_data = scaler.fit_transform(df)

     df = pd.DataFrame(normalized_data, columns=df.columns)

     print("Normalized Dataset: ")
     print(df)
```

```
Normalized Dataset:
            ph  Hardness    Solids  Chloramines   Sulfate  Conductivity  \
0     0.501200  0.561805  0.427759     0.524547  0.661087      0.779857
1     0.117052  0.163786  0.382596     0.453558  0.500481      0.841303
2     0.617462  0.663836  0.409333     0.735459  0.500481      0.466566
3     0.642310  0.611819  0.453401     0.605586  0.607320      0.347575
4     0.730844  0.436341  0.368991     0.444095  0.391193      0.423142
...        ...       ...       ...          ...       ...           ...
3123  0.225744  0.502690  0.987569     0.510287  0.621478      0.698398
3124  0.584322  0.502012  0.355426     0.605803  0.500481      0.410324
3125  0.768210  0.408184  0.686129     0.529887  0.500481      0.495462
3126  0.278109  0.697418  0.243714     0.418127  0.500481      0.432758
3127  0.591836  0.510182  0.356980     0.546868  0.500481      0.270582

      Organic_carbon  Trihalomethanes  Turbidity  Potability
0           0.306829         0.722415   0.288245         0.0
1           0.551982         0.395786   0.623577         0.0
2           0.638222         0.503282   0.308484         0.0
3           0.718296         0.864635   0.651519         0.0
4           0.367016         0.136596   0.530758         0.0
...              ...              ...        ...         ...
3123        0.486325         0.506132   0.609436         1.0
3124        0.793202         0.503028   0.252282         1.0
3125        0.340500         0.539770   0.361469         1.0
3126        0.347133         0.621186   0.668942         1.0
3127        0.601028         0.634078   0.145611         1.0
```

Data Normalization Code

```
[63] from sklearn.model_selection import train_test_split

     # Train:Validation:Test = 75:10:15
     X_train, X_temp, y_train, y_temp = train_test_split(
         X_scaled, y, test_size=0.25, random_state=42, stratify=y)

     X_val, X_test, y_val, y_test = train_test_split(
         X_temp, y_temp, test_size=0.6, random_state=42, stratify=y_temp)

     print("Train set size:", X_train.shape, y_train.shape)
     print("Validation set size:", X_val.shape, y_val.shape)
     print("Test set size:", X_test.shape, y_test.shape)
     print("\nTrain set class distribution:\n", y_train.value_counts())
     print("\nValidation set class distribution:\n", y_val.value_counts())
     print("\nTest set class distribution:\n", y_test.value_counts())
```

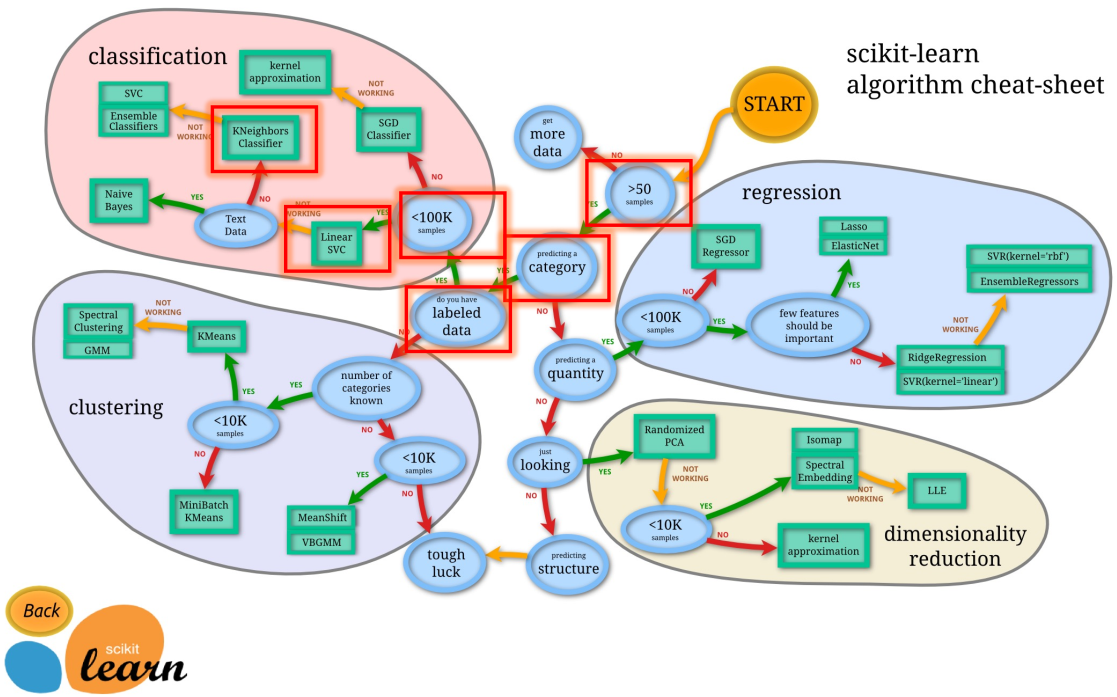
```
Train set size: (2346, 9) (2346,)
Validation set size: (312, 9) (312,)
Test set size: (470, 9) (470,)

Train set class distribution:
 Potability
0    1447
1     899
Name: count, dtype: int64

Validation set class distribution:
 Potability
0    193
1    119
Name: count, dtype: int64
```

Data Splitting Code

V. Updating the model

The Most Suitable Models for this Project

- Logistic Regression
- SVC / SVM
- Randomforest



KOREA UNIVERSITY

V. Updating the model

1) Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Initialization
lr_model = LogisticRegression(max_iter=1000)

# Model training
lr_model.fit(X_train, y_train)
```

Training code

```
# Validation dataset prediction
y_val_pred = lr_model.predict(X_val)
val_accuracy = accuracy_score(y_val, y_val_pred)
print(f'Validation Accuracy: {val_accuracy:.4f}')
print('Classification Report:')
print(classification_report(y_val, y_val_pred))
print('Confusion Matrix:')
print(confusion_matrix(y_val, y_val_pred))
```

```
Validation Accuracy: 0.6186
Classification Report:
              precision    recall  f1-score   support

           0       0.62      1.00      0.76       193
           1       0.00      0.00      0.00       119

    accuracy                           0.62       312
   macro avg       0.31      0.50      0.38       312
weighted avg       0.38      0.62      0.47       312

Confusion Matrix:
[[193   0]
 [119   0]]
```

Validation set score: 0.6186

KOREA UNIVERSITY

V. Updating the model

2) SVC / SVM

```
from sklearn.svm import SVC

svm_model = SVC(kernel='rbf', probability=True, random_state=42)

svm_model.fit(X_train, y_train)

y_val_pred = svm_model.predict(X_val)
val_accuracy = accuracy_score(y_val, y_val_pred)
print(f'Validation Accuracy: {val_accuracy:.4f}')
print('Classification Report:')
print(classification_report(y_val, y_val_pred))
print('Confusion Matrix:')
print(confusion_matrix(y_val, y_val_pred))
```

```
Validation Accuracy: 0.6731
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.93      0.78       193
           1       0.70      0.25      0.37       119

    accuracy                           0.67       312
   macro avg       0.68      0.59      0.57       312
weighted avg       0.68      0.67      0.62       312

Confusion Matrix:
[[180  13]
 [ 89  30]]
```

Training & Validation score code

```
Test Accuracy: 0.6681
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.79      0.75       290
           1       0.58      0.48      0.52       180

    accuracy                           0.67       470
   macro avg       0.64      0.63      0.63       470
weighted avg       0.66      0.67      0.66       470

Confusion Matrix:
[[228  62]
 [ 94  86]]
```

Test set score: 0.6681

V. Updating the model

3) Randomforest

```python
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')

rf_model.fit(X_train, y_train)

y_val_pred = rf_model.predict(X_val)
val_accuracy = accuracy_score(y_val, y_val_pred)
print(f'Validation Accuracy: {val_accuracy:.4f}')
print('Classification Report:')
print(classification_report(y_val, y_val_pred))
print('Confusion Matrix:')
print(confusion_matrix(y_val, y_val_pred))
```

```
Validation Accuracy: 0.6667
Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.91      0.77       193
           1       0.65      0.27      0.38       119

    accuracy                           0.67       312
   macro avg       0.66      0.59      0.58       312
weighted avg       0.66      0.67      0.62       312

Confusion Matrix:
[[176  17]
 [ 87  32]]
```

Training & Validation score code

```python
y_test_pred = rf_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_test_pred)
print(f'Test Accuracy: {test_accuracy:.4f}')
print('Classification Report:')
print(classification_report(y_test, y_test_pred))
print('Confusion Matrix:')
print(confusion_matrix(y_test, y_test_pred))
```

```
Test Accuracy: 0.6617
Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.92      0.77       290
           1       0.65      0.25      0.36       180

    accuracy                           0.66       470
   macro avg       0.66      0.58      0.57       470
weighted avg       0.66      0.66      0.61       470

Confusion Matrix:
[[266  24]
 [135  45]]
```

Test set score: 0.6617

VI. Result

[Original Code]

1) Dataset
- Train: 1,608
- Validation: 201
- Test: 202

2) Machine Learning Model
- Xgboost

3) Performance
- Accuracy : 0.6584

[Updated Code]

1) Dataset
- Train: 2,346
- Validation: 312
- Test: 470

2) Machine Learning Model
- Logistic Regression
- SVC / SVM
- Randomforest

3) Performance
- Highest Accuracy : 0.6681

KOREA UNIVERSITY

Q & A

E-mail : kimwc620@korea.ac.kr

GitHub address :

https://github.com/SkyDreamer14/WaterPotability