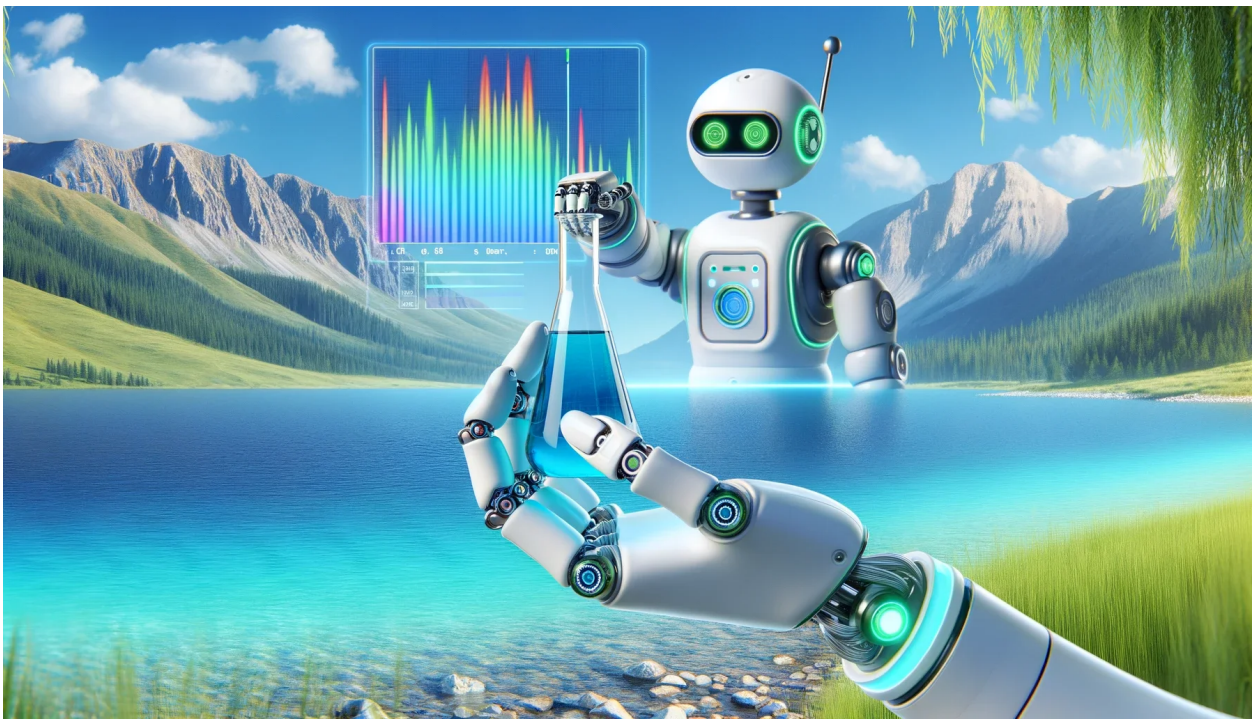


Water Potability Prediction

EECS 3401 A | Introduction to Artificial Intelligence

Professor Dr. Ruba Alomari | York University

Aakanksha Verma (218124156),
Hulya Yasar (219068568), and
Franco Miguel Dela Cruz (216413742)

21/11/2023

—

Table of Contents

Introduction

The goal of this project is to perform exploratory data analysis (EDA), prepare the data for modelling, train and evaluate the data through various machine learning models, and present the results.

We have chosen a dataset from Kaggle which predicts water potability using various water quality attributes.

Framing the Problem and Looking at the Big Picture

Frame the Problem

The dataset selected is a classification task, which means there is a category picked, in this case, if the water is potable or not. Through the use of supervised learning, meaning that the training examples are labelled, and batch learning, because of a small dataset, no continuous flow of data coming into the system and no need to adjust to changing data rapidly, the problem will be solved.

The objective is to find a model that allows us to predict whether the water in a river is drinkable or not to know if we should invest in the development of a sanitation network of water.

This task will therefore be binary classification, we will assign in the remainder of this project the value 1 for “Drinkable” and 0 for “Non-Drinkable”

Look at the big picture

Predictions will be used to help inform people if the body of water is drinkable. This will be done through the following properties of the water: pH value, hardness, solids, chloramines, sulfate, conductivity, organic carbon, trihalomethanes, and turbidity.

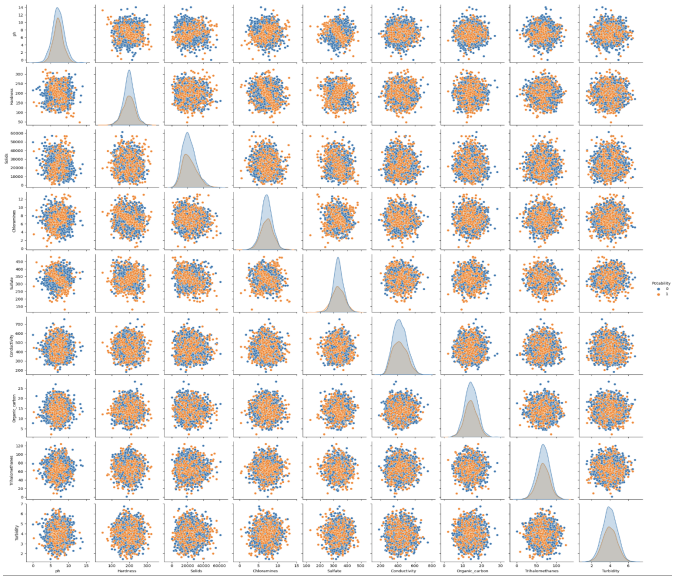
Description of the Dataset

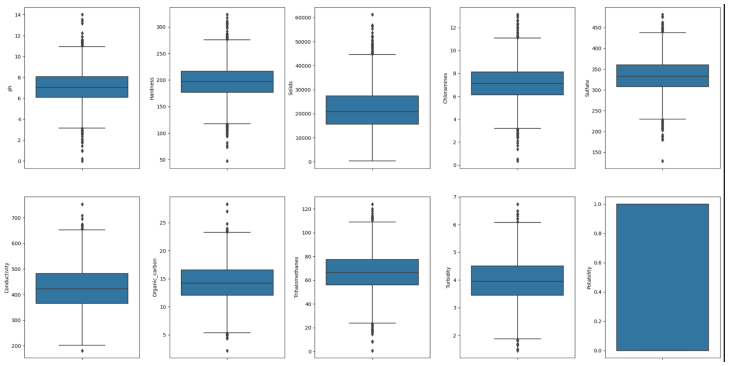
The dataset for this project focuses on predicting the potability of water, where potability is represented as a categorical variable with values of 0 or 1. The primary goal is to develop a dependable prediction model based on nine numerical features. Each feature is of type float, indicating a numeric nature, and contributes distinct information to the analysis. Notably, the features exhibit variations in scale, reflecting their diverse measurement units.

EDA Graphs

1. Pairplot

The pairplot suggests a lack of apparent correlation between the features in the dataset. The points, coloured based on whether the water is potable or not, are kind of all over the place in the scatter plots. This observation indicates that the features, as visualized in pairs, do not exhibit strong linear relationships or dependencies.



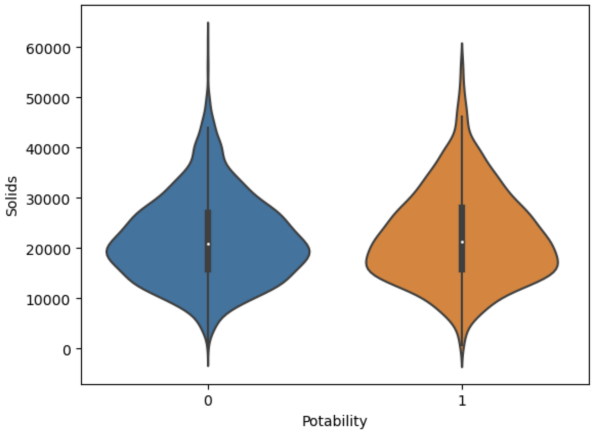


2. Boxplots

Boxplots illustrate the distributions of features related to water quality and potability. Differences in means, standard deviations, and the prevalence of outliers are evident between potable and non-potable water samples. This will therefore lead us to scale our data using a StandardScaler. We intentionally keep the unusual data points, known as outliers, because they are important for training a model that can tell the difference between good and bad water quality.

3. Violinplot

The Violinplot graph depicting Potability and Solids showcases the distribution of Solids content in water concerning its potability status. The width of the violin plots represents the density of data points, with broader sections indicating higher concentration. For Potable water, the plot shows a wider section at lower Solid values, while for Non-Potable water, it extends towards higher Solid values. This suggests that Solids content might be a relevant factor in distinguishing between potable and non-potable water.



Data Cleaning and Processing

Data Cleaning

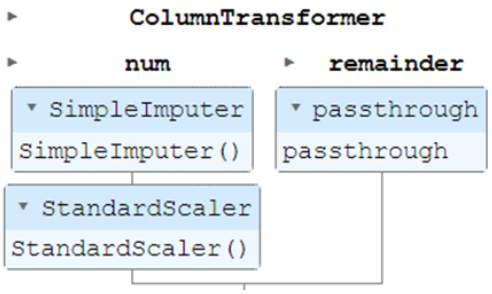
ph	491
Hardness	0
Solids	0
Chloramines	0
Sulfate	781
Conductivity	0
Organic_carbon	0
Trihalomethanes	162
Turbidity	0
Potability	0
dtype: int64	

In this process, duplicate and missing values were looked for in the dataset. There were no duplicates, but there were some missing values for the columns *'ph'*, *'sulfate'* and '*trihalomethanes*'.
We decided to try two ways to handle these missing values. The first way is to delete the '*ph*' and '*sulfate*' columns and replace the missing values in '*trihalomethanes*' with the mean value of the column. This is because '*ph*' and '*sulfate*' have the most missing values relative to the dataset. The second way is to replace all missing values with the mean values of their respective columns. A copy was made of the original dataset so that one can be used for the removed columns way (called *water_removedColumns* or *wrc*) and the other used with only the substitution of mean values (called *water*).

Processing

There are two pipelines created to handle the different cleaning strategies. As there are no categorical (object data type) columns in our dataset we have only created the numerical

columns in the pipeline along with the remainder which contains our target value of '*Potability*'. In the *SimpleImputer()* parameter, we have used the *strategy* of '*mean*' but as that is the default for the parameter it is not shown in the image. Lastly, the pipeline for both ways that we will be testing looks the same as they have been created the same way, just using a different variable, so they will work for their respective purposes.



The pipelines are also fitted to their respective datasets so that they are configured correctly for future training and testing. By this, we mean that the *wrc_pipeline* is fitted to the *wrc* dataset and the *water_pipeline* is fitted to the *water* dataset.

Training and Evaluation of Three machine learning Algorithms, Analyze Findings, and Compare Results

Model Selection and Hyperparameter Tuning

In this step, three different classification algorithms were selected to model the water potability problem: Support Vector Machine (SVM), Random Forest, and Gradient Boosting. These algorithms were chosen for their effectiveness in handling classification tasks and capturing complex relationships within the data.

Hyperparameter tuning is important to create the best model and was performed using GridSearchCV. It systematically explores the predefined parameters and selects the combination that maximizes the model's performance.

SVM: The tuned hyperparameters for this model include kernel type ('rbf'), the regularization parameter 'C', and the 'gamma' parameter.
Random Forest: For the Random Forest model, hyperparameters such as the splitting criterion ('gini' or 'entropy'), the maximum depth of the trees, and the random state were explored.
Gradient Boosting: The Gradient Boosting algorithm was tuned by varying the criterion for splits ('friedman_mse' or 'squared_error'), maximum tree depth, and random state.

For each of the algorithms, the chosen models were tested on a validation set to gauge their generalization performance after the hyperparameters were adjusted. The evaluation measures were F1-score, recall, accuracy, and precision. These metrics offer a thorough grasp of how well the models can distinguish between samples of potable and non-potable water.

Model Evaluation

The top-performing models were then trained/tested on a different set that they had not faced during training or hyperparameter tuning after being evaluated on the validation set. To make sure the models can effectively generalize to new, untested data, this phase is essential.

When conducting the training of data with the values retrieved from the grid search, there were slight changes made when doing the training for the dataset with deleted columns. The changes were in the Random Forest and Gradient Boosting max depth value, which were put higher than what was suggested in the grid search. The reason for this was that even though the accuracy may have been a bit lower than expected, the precision, recall, and F1-score were more balanced out.

Overall, each of the final models had given similar results in the evaluation metrics. For

the deleted columns dataset, accuracy ranged from 60 to 63 % for the models, whereas it ranged from 67 to 68 % in the mean-filled dataset. The precision, recall and F1-score of the models were also similar and balanced out.

Cross-Validation

Before deciding on the final best model for the dataset, further validation of the robustness of the models was done using k-fold cross-validation. This method involves dividing the training dataset into k subset (fold), training the model k times, and evaluating the performance on the different folds each time. It helps provide more reliable performance estimates by reducing the manual need to continually split the dataset into training and testing sets.

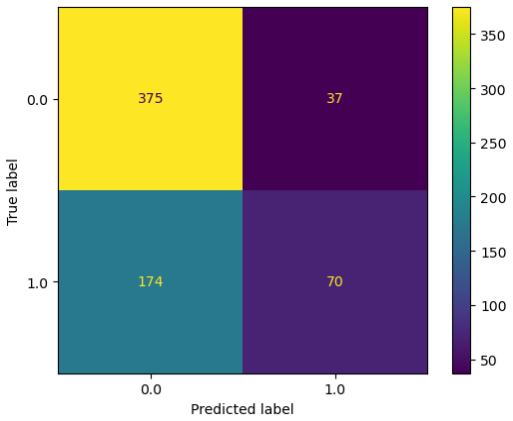
Once the values for each cross-validation were retrieved, the mean accuracy of all folds was calculated to get an overall assessment of the models' performance.

Model Comparison and Selection

The models were compared based on their mean cross-validation accuracy. The results of the deleted columns dataset, SVM had the best results of about 61% accuracy, whereas the mean-filled dataset had Random Forest as the best result with 67% accuracy. Comparing these results, the model with the highest mean accuracy was selected as the final model for predicting water potability, meaning the Random Forest model of the mean-filled dataset.

Graphs of Best Performing Algorithm

Confusion Matrix



A confusion matrix contrasts a model's results with the real world to visually represent the performance of the model in terms of true positives, true negatives, false positives, and false negatives.

- **Precision for Class 1:** Precision is the number of genuine positives divided by the total number of anticipated positives. To measure how well this classifier does at making positive predictions, divide 81 by (81 + 51).
- **Recall for Class 1**: It is calculated by dividing the number of correct identifications by the total number of positive results. In this case, the model's accuracy in identifying the positive class can be expressed as 81 / (81 + 163).
- **F1 Score for Class 1:** The F1 score is the harmonic mean of precision and recall. It is the ratio between precision and recall, and it is computed as 2 * (Precision * Recall) / (Precision + Recall).
- **Specificity:** The genuine negative rate is found by dividing TN by (TN + FP). It's a test of how well the model can identify the undesirable category.
- **Class Imbalance:** The positive and negative classes appear 11 to exist, with 412 examples of the former and 244 of the latter. This may necessitate resampling to equalize the classes or modifying the classification threshold to improve the model's performance.

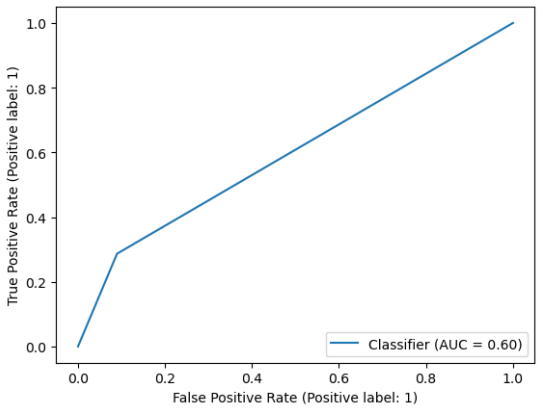
Overall, it is seen that:

- A larger ratio of true negatives to true positives suggests that the model is more accurate at predicting the negative class than the positive class.
- With such a high rate of false negatives, it seems likely that the model is missing some genuine positives.

Due to the large number of false positives and false negatives relative to the number of actual positives, the accuracy and recall for the positive class are likely to be subpar. According to the matrix, the model detects negative classes well but could improve positive class detection.

ROC-AUC Curve

A Receiver Operating Characteristic (ROC) curve illustrates the relationship between a model's true positive rate and false positive rate, based on different cut-off thresholds, and provides an understanding of a model's ability to distinguish between them.



- **AUC (Area Under Curve) Value:** The AUC value, representing the Area Under the Curve, is 0.60. This indicates that the classifier's capability to differentiate between the positive and negative classifications is marginally superior to a random prediction. An AUC value of 0.50 indicates a classifier that performs randomly, while an AUC value of 0.60 suggests a moderate level of predictive ability, but not a high one.
- **Curve Shape:** The ROC curve is ideally expected to exhibit a concave shape towards the upper left corner of the plot, which signifies a high true positive rate and a low false positive rate. The curve closely approximates the diagonal line, indicating that the classifier is not highly proficient in differentiating between the two classes.
- **True Positive Rate:** The true positive rate has a positive correlation with the false positive rate, as expected. However, there is no significant spike observed, suggesting that the classifier is not efficiently identifying the genuine positives.
- **False Positive Rate:** The rate of false positives exhibits a consistent increase, indicating that as the classifier attempts to identify more genuine positives, it also mistakenly categorizes negative instances as positive.

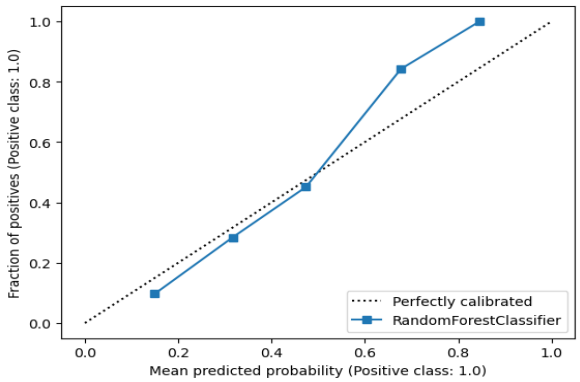
This classifier has an AUC of 0.60, higher than random chance but not particularly accurate. Additions, classification techniques, or model parameters may improve this model.

Calibration Curve

Calibration curves are a helpful visual aid to understand how well a classifier is calibrated, which means indicating how well the predicted probabilities align with the actual outcomes.

- **Linearity:** The dotted line shows a completely calibrated model with matching

probabilities and outcomes. The calibration curve is the solid line with squares, representing the Random Forest Classifier. It should be on the dotted line for correct calibration.



- **Calibration:** The Random Forest Classifier's line is near to the dotted line, indicating good model probability. However, variations, notably in the lower and higher probability end, show that the model may be underconfident in its extreme predictions.
- **Resolution and refinement:** The graph illustrates that the model has a good resolution, as shown by the squares across projected probabilities. The square size may indicate the number of samples in the projected probability bin.
- **Confidence:** Lack of error bars or confidence intervals around the squares makes it difficult to assess uncertainty in these estimates. Narrower confidence intervals usually indicate higher calibration estimate confidence.

The calibration curve reveals that the Random Forest Classifier is well-calibrated. Probabilistic predictions require calibration when model outputs have major repercussions and depend on a reliable probability estimate.

Limitations

1. **Imbalanced Dataset:** The dataset exhibits a class imbalance concerning water potability. It is a 60 to 40 balance between not potable water and potable water respectively. This may look somewhat alright, but with the majority of samples falling into one class, it potentially leads to biased model predictions.
2. **Assumption of Feature Independence:** The current analysis assumes independence between features. However, in real-world scenarios, features may exhibit correlations or interactions that are not captured by the models used.
3. **GridSearch Limited Hyperparameter Tuning:** When conducting a grid search for the three algorithms, there were only about three parameters used to find the best parameter values. The issue came that when adding more parameters with possible values, the search was taking extremely long, above 30 minutes, which caused us to limit the number of parameters we looked into.

Conclusion

Through exploring artificial intelligence by uncovering data cleaning and processing, training and evaluating machine learning algorithms, and reading datasets combined with graphing best-performing algorithms, we have analyzed our selected dataset. From this, we have selected a random forest model as the best model.

Appendix 1 - Notebook

Project: Exploring and Predicting with a Dataset

Author: Aakanksha Verma, Hulya Yasar, Miguel Dela Cruz
Course: LE/EECS 3401 A | Introduction to Artificial Intelligence and Logic Programming

Original Dataset Source: Kadiwal, Aditya. (2020). Water Quality. Kaggle.
https://www.kaggle.com/datasets/adityakadiwal/water-potability.

Uploaded and used from:
https://raw.githubusercontent.com/Verma-Aakanksha/EECS3401-Project/main/water_potability.csv

Water Quality Dataset Description

Attributes for dataset: The below attributes are copied **AS IS** from the original dataset.

1. *pH value:* PH is an important parameter in evaluating the acid–base balance of water. It is also the indicator of acidic or alkaline condition of water status. WHO has recommended maximum permissible limit of pH from 6.5 to 8.5. The current investigation ranges were 6.52–6.83 which are in the range of WHO standards.
2. *Hardness:* Hardness is mainly caused by calcium and magnesium salts. These salts are dissolved from geologic deposits through which water travels. The length of time water is in contact with hardness producing material helps determine how much hardness there is in raw water. Hardness was originally defined as the capacity of water to precipitate soap caused by Calcium and Magnesium.
3. *Solids (Total dissolved solids - TDS):* Water has the ability to dissolve a wide range of inorganic and some organic minerals or salts such as potassium, calcium, sodium, bicarbonates, chlorides, magnesium, sulfates etc. These minerals produced un-wanted taste and diluted color in appearance of water. This is the important parameter for the use of water. The water with high TDS value indicates that water is highly mineralized. Desirable limit for TDS is 500 mg/l and maximum limit is 1000 mg/l which prescribed for drinking purpose.
4. *Chloramines:* Chlorine and chloramine are the major disinfectants used in public water systems. Chloramines are most commonly formed when ammonia is added to chlorine to treat drinking water. Chlorine levels up to 4 milligrams per liter (mg/L or 4 parts per million (ppm)) are considered safe in drinking water.
5. *Sulfate:* Sulfates are naturally occurring substances that are found in minerals, soil, and rocks. They are present in ambient air, groundwater, plants, and food. The principal commercial use of sulfate is in the chemical industry. Sulfate concentration in seawater is about 2,700 milligrams per liter (mg/L). It ranges from 3 to 30 mg/L in most

freshwater supplies, although much higher concentrations (1000 mg/L) are found in some geographic locations.

6. *Conductivity:* Pure water is not a good conductor of electric current rather's a good insulator. Increase in ions concentration enhances the electrical conductivity of water. Generally, the amount of dissolved solids in water determines the electrical conductivity. Electrical conductivity (EC) actually measures the ionic process of a solution that enables it to transmit current. According to WHO standards, EC value should not exceeded 400 µS/cm.
7. *Organic_carbon:* Total Organic Carbon (TOC) in source waters comes from decaying natural organic matter (NOM) as well as synthetic sources. TOC is a measure of the total amount of carbon in organic compounds in pure water. According to US EPA < 2 mg/L as TOC in treated / drinking water, and < 4 mg/Lit in source water which is use for treatment.
8. *Trihalomethanes:* THMs are chemicals which may be found in water treated with chlorine. The concentration of THMs in drinking water varies according to the level of organic material in the water, the amount of chlorine required to treat the water, and the temperature of the water that is being treated. THM levels up to 80 ppm is considered safe in drinking water.
9. *Turbidity:* The turbidity of water depends on the quantity of solid matter present in the suspended state. It is a measure of light emitting properties of water and the test is used to indicate the quality of waste discharge with respect to colloidal matter. The mean turbidity value obtained for Wondo Genet Campus (0.98 NTU) is lower than the WHO recommended value of 5.00 NTU.
10. *Potability:* Indicates if water is safe for human consumption where 1 means Potable and 0 means Not potable.

Missing values:

- ph: 491
- Sulfate: 781
- Trihalomethanes: 162

Duplicated values: There are no duplicate values in the dataset.

1: Look at the Big Picture and Frame the Problem.

1.1 Frame the Problem

1. Supervised learning – training examples are labelled.
2. A classification task – predict a category.
3. Batch learning
    - Small data set
    - No continuous flow of data coming into the system
    - No need to adjust to changing data rapidly

1.2 Look at the big picture

Predictions will be used to help inform people if the body of water is drinkable. This will be done through the following properties of the water: pH value, hardness, solids, chloramines, sulfate, conductivity, organic carbon, trihalomethanes, and turbidity.

2: Description of the Dataset and Graphs of EDA.

In [ ]:

```
# Import libraries
import sklearn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:

```
# Load the dataset
url =
"https://raw.githubusercontent.com/Verma-Aakanksha/EECS3401-Project/main/water_potability.csv"
water = pd.read_csv(url, sep=',')

# Backup copy of the dataset
water_backup = water
```

2.1 Take a Quick Look at the Data Structure

2.1.1 Examine the number of rows, the number of columns, and columns labels in the dataset.

In [ ]: `water`

2.1.2 Looking at the First 5 Rows of the Dataset

In [ ]: `water.head()`

2.1.3 Summary of the Numerical Data in the Dataset

In [ ]: `water.describe()`

2.1.4 Description of the Data

Total number of rows, each attribute's type, and the number of non-null values.

In [ ]: `water.info()`

2.1.5 Number of Rows and Columns in the Dataset

In [ ]:
```
water.shape
```

2.1.6 Showing Count of Target Value

In [ ]:
```
water["Potability"].value_counts()
```

2.2 EDA Graphs to Explore and Visualize the Data to Gain Insights.

2.2.1 Histogram of the Data

In [ ]:
```
water.hist(figsize=(24, 16))
plt.show()
```

2.2.2 Correlations Between the Features

In [ ]:
```
# Check for correlation between attributes using sns.pairplot.
sns.pairplot(water, hue="Potability")
```

2.2.2.1 Correlations using Pearson correlation coefficient.

In [ ]:
```
#corr method has pearson standard correlation coefficient as the default
corr_matrix = water.corr(numeric_only=True)
corr_matrix
```

Let's look at correlations with regard to our target

In [ ]:
```
corr_matrix["Potability"].sort_values(ascending=False)
```

2.2.3 Boxplot to Visualize Dataset and Checking Outliers

In [ ]:
```
# Referenced from
https://www.kaggle.com/code/imakash3011/water-quality-prediction-7-model?s
criptVersionId=72225152&cellId=22
# Visualizing dataset and also checking for outliers
fig, ax = plt.subplots(ncols = 5, nrows = 2, figsize = (20, 10))
index = 0
ax = ax.flatten()

for col, value in water.items():
    sns.boxplot(y=col, data=water, ax=ax[index])
    index += 1
plt.tight_layout(pad = 0.5, w_pad=0.7, h_pad=5.0)
```

2.2.4 Violinplot Graph for Comparison of Potability and Solids

Potatbility because it is the target, and Solids because it is highest in corr_matrix against Potatability.

In [ ]:

```
#violinplot graph for comparison of Potability and Solids
sns.violinplot(x='Potability',y='Solids',data=water,hue='Potability')
```

2.2.5 Barpot Grapgh for Comparison of Potability and Solids

Potatbility because it is the target, and Solids because it is highest in corr_matrix against Potatability

In [ ]:

```
#barpot grapgh for comparison of Potability and Solids
plt.figure(figsize=(8, 6))
sns.barplot(x='Potability', y='Solids', data=water, estimator=sum,
errorbar=None)
plt.xlabel('Potability')
plt.ylabel('Total Solids')
plt.title('Bar Plot of Potability with respect to Total Solids')
plt.show()
```

2.2.6 Pie Chart Using the Value Counts of the 'Potability' Column

In [ ]:

```
# Create a pie chart using the value counts of the 'Potability' column
plt.pie(water['Potability'].value_counts(),labels =
list(water['Potability'].unique()),autopct="%0.1f%%" )
# Show the pie chart
plt.show()
```

3: Data Cleaning and Preprocessing

3.1 Data Cleaning: Duplicate and Missing Values and Dealing with Them

3.1.1 Check for duplicate rows and remove them if any.

In [ ]:

```
# Check for duplicate rows and delete them
water.duplicated().sum()
```

No duplicates, so nothing else is needed for this part.

3.1.2 Handling the missing values

In [ ]:

```
# Find the number of missing values in each column
water.isna().sum()
```

There are 2 ways that we will look at the data now.

1. We will delete both 'ph' and 'Sulfate' columns and have 'Trihalomethanes' missing values filled with mean values.
2. We will fill in the mean values for all missing values.

1. Deleting 'ph' and 'Sulfate' Columns

In [ ]:

```
# Coping the water dataset into another variable to do calculations with
the removed columns.
water_removedColumns = water.copy()
water_removedColumns.drop(labels=['ph','Sulfate'], axis=1, inplace=True)
```

In [ ]:

```
# 'ph' and 'Sulfate' columns have been removed, checking info to verify
water_removedColumns.info()
```

In [ ]:

```
# Checking that only 'Trihalomethanes' is the only column with missing
values in water_removedColumns
water_removedColumns.isna().sum()
```

In [ ]:

```
# Checking that water still has all 3 columns with missing values.
water.isna().sum()
```

For the remaining missing values, we will fill them with the mean value. We will do this by creating a pipeline, that will also scale the features and perform encoding in the next step.

Ways 1 and 2 will be now shown in the pipeline as the data is now cleaned up according to their requirements.

3.2 Processing: Create a Pipeline and Apply it to Data

In [ ]:

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
```

1. Deleting 'ph' and 'Sulfate' Columns

In [ ]:

```
# Create the num columns, we do not have any object data types so no
categorical columns are needed.
num_cols_wrc =
water_removedColumns.select_dtypes(include='number').columns.to_list()

# Exclude the target from numerical columns
num_cols_wrc.remove("Potability")

# Create pipelines for numeric and categorical columns
num_pipeline_wrc = make_pipeline(SimpleImputer(strategy='mean'),
StandardScaler())

# Use ColumnTransformer to set the estimators and transformations
preprocessing_wrc = ColumnTransformer([('num', num_pipeline_wrc,
num_cols_wrc)], remainder='passthrough')
```

In [ ]:

```
# Columns in the num columns of the pipeline for water_removedColumns
num_cols_wrc
```

Display Water_RemovedColumns Pipeline

In [ ]:

```
preprocessing_wrc
```

Applying Processing Pipeline

In [ ]:

```
wrc_prepared = preprocessing_wrc.fit_transform(water_removedColumns)

# Scikit-learn strips the column headers, so adding them back
wrc_feature_names = preprocessing_wrc.get_feature_names_out()
wrc_prepared = pd.DataFrame(data = wrc_prepared, columns =
wrc_feature_names)

# Display the prepared water_removedColumns data
wrc_prepared
```

In [ ]:

```
# Columns in the prepared pipeline for water_removedColumns
wrc_prepared.columns
```

2. Fill in Mean Values for All Missing Values

In [ ]:

```
# Create the num columns, we do not have any object data types so no
```

```
categorical columns are needed.
num_cols = water.select_dtypes(include='number').columns.to_list()

# Exclude the target from numerical columns
num_cols.remove("Potability")

# Create pipelines for numeric and categorical columns
num_pipeline = make_pipeline(SimpleImputer(strategy='mean'),
StandardScaler())

# Use ColumnTransformer to set the estimators and transformations
preprocessing = ColumnTransformer([('num', num_pipeline, num_cols)],
remainder='passthrough' )
```

In [ ]:

```
# Columns in the num columns of the pipeline for water
num_cols
```

Display Water Pipeline

In [ ]:
```
preprocessing
```

Apply Processing Pipeline

In [ ]:

```
water_prepared = preprocessing.fit_transform(water)

# Scikit-learn strips the column headers, so adding them back
feature_names=preprocessing.get_feature_names_out()
water_prepared = pd.DataFrame(data=water_prepared, columns=feature_names)

# Display the prepared water data
water_prepared
```

In [ ]:

```
# Columns of the prepared pipeline for water
water_prepared.columns
```

4. Training and Evaluation of Three Machine Learning Algorithms, Analyzing Findings, and Comparing Results

1. Deleting 'ph' and 'Sulfate' Columns

In [ ]:

```
# Separating target from the rest of the data
X_wrc = wrc_prepared.drop(["remainder__Potability"], axis=1)
y_wrc = wrc_prepared["remainder__Potability"]
```

Using GridSearchCV to Find the Best Value for Parameters for SVC, Random Forest, and Gradient Boosting Algorithms

Split the dataset into 60% training, 20% validation, and 20% testing.

In [ ]:

```
from sklearn.model_selection import train_test_split

X_wrc_gs_train, X_wrc_validation_test, y_wrc_gs_train,
y_wrc_validation_test = train_test_split(X_wrc, y_wrc, test_size=0.4,
random_state=42)

X_wrc_validation, X_wrc_gs_test, y_wrc_validation, y_wrc_gs_test =
train_test_split(X_wrc_validation_test, y_wrc_validation_test,
test_size=0.5, random_state=42)

print(X_wrc_gs_train.shape, y_wrc_gs_train.shape, X_wrc_validation.shape,
y_wrc_validation.shape, X_wrc_gs_test.shape, y_wrc_gs_test.shape)
```

GridSearch for SVM

In [ ]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

# code author luisguiserrano

wrc_svm_parameters = {
                      'kernel': ['rbf'],
                      'C': [0.01, 0.1, 1 , 10],
                      'gamma': [0.01, 1, 10]
                     }
wrc_svm = SVC()
wrc_svm_gs = GridSearchCV(estimator = wrc_svm, param_grid =
wrc_svm_parameters)
wrc_svm_gs.fit(X_wrc_gs_train, y_wrc_gs_train.values.ravel())

# What is the best estimator from the GridSearch above
wrc_svm_winner = wrc_svm_gs.best_estimator_

# What is the accuracy of the best estimator
wrc_svm_winner.score(X_wrc_validation, y_wrc_validation)
```

In [ ]:

```
# Display the best estimator
wrc_svm_winner
```

GridSearch for Random Forest

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier

wrc_rfc_parameters = {
                        'criterion': ['gini', 'entropy'],
                        'max_depth': [2, 4, 6, 8, 10, 13],
                        'random_state':[42]
                      }
wrc_rfc = RandomForestClassifier()
wrc_rfc_gs = GridSearchCV(estimator = wrc_rfc, param_grid =
wrc_rfc_parameters)
wrc_rfc_gs.fit(X_wrc_gs_train, y_wrc_gs_train.values.ravel())

# What is the best estimator from the GridSearch above
wrc_rfc_winner = wrc_rfc_gs.best_estimator_

# What is the accuracy of the best estimator
wrc_rfc_winner.score(X_wrc_validation, y_wrc_validation)
```

In [ ]:

```
# Display the best estimator
wrc_rfc_winner
```

GridSearch for Gradient Boosting

In [ ]:

```
from sklearn.ensemble import GradientBoostingClassifier

wrc_gbc_parameters = {
                       'criterion': ['friedman_mse', 'squared_error'],
                       'max_depth': [2, 4, 6, 8, 10],
                       'random_state': [42]
                      }
wrc_gbc = GradientBoostingClassifier()
wrc_gbc_gs = GridSearchCV(estimator = wrc_gbc, param_grid =
wrc_gbc_parameters)
wrc_gbc_gs.fit(X_wrc_gs_train, y_wrc_gs_train.values.ravel())

# What is the best estimator from the GridSearch above
wrc_gbc_winner = wrc_gbc_gs.best_estimator_
```

```
# What is the accuracy of the best estimator
wrc_gbc_winner.score(X_wrc_validation, y_wrc_validation)
```

In [ ]:

```
# Display the best estimator
wrc_gbc_winner
```

Split the dataset into a training dataset (80%) and testing dataset.

In [ ]:

```
X_wrc_train, X_wrc_test, y_wrc_train, y_wrc_test = train_test_split(X_wrc,
y_wrc, test_size=0.2, random_state=42)
print(X_wrc_train.shape, y_wrc_train.shape, X_wrc_test.shape,
y_wrc_test.shape)
```

Using Values Retrieved in GridSearch to Train Data

Train a SVM Model (SVC)

In [ ]:

```
# Training SVM Model
wrc_svm_model = SVC(kernel='rbf', C=10, gamma=0.01)
wrc_svm_model.fit(X_wrc_train, y_wrc_train.values.ravel())
wrc_svm_model
```

Train a Random Forest Model

In [ ]:

```
# Training Random Forest Model
wrc_random_forest_model = RandomForestClassifier(max_depth=13,
random_state=42)
wrc_random_forest_model.fit(X_wrc_train, y_wrc_train)
wrc_random_forest_model
```

Train a Gradient Boosting Model

In [ ]:

```
# Training Gradient Boosting Model
wrc_gradient_boosting_model =
GradientBoostingClassifier(criterion='squared_error', max_depth=6,
random_state=42)
wrc_gradient_boosting_model.fit(X_wrc_train, y_wrc_train)
wrc_gradient_boosting_model
```

Test Models on the X_wrc_Test

In [ ]:

```
# Testing the model
```

```
wrc_svm_y_predict = wrc_svm_model.predict(X_wrc_test)
wrc_random_forest_y_predict = wrc_random_forest_model.predict(X_wrc_test)
wrc_gradient_boosting_y_predict =
wrc_gradient_boosting_model.predict(X_wrc_test)
```

Report the Classification_Report and Accuracy of Each Model on the y_wrc_test and y_wrc_predict.

In [ ]:

```
from sklearn.metrics import classification_report, accuracy_score

# Evaluate model performances
wrc_models = [wrc_svm_model, wrc_random_forest_model,
wrc_gradient_boosting_model]
wrc_predicts = [wrc_svm_y_predict, wrc_random_forest_y_predict,
wrc_gradient_boosting_y_predict]
wrc_model_names = ['SVM', 'Random Forest', 'Gradient Boosting']

for model, predict, name in zip(wrc_models, wrc_predicts,
wrc_model_names):
    accuracy = accuracy_score(y_wrc_test, predict)
    print(f"------ {name} ------")
    print(f"Accuracy: {accuracy}")
    print("Classification Report:")
    print(classification_report(y_wrc_test, predict))
```

Train Models Using KFold Cross-Validation with 5 folds

In [ ]:

```
from sklearn.model_selection import cross_val_score
# cross-validation for svm model
wrc_svm_score = cross_val_score(wrc_svm_model, X_wrc_train, y_wrc_train,
cv=5)
wrc_svm_score
```

In [ ]:

```
# cross-validation for random-forest model
wrc_rfc_score = cross_val_score(wrc_random_forest_model, X_wrc_train,
y_wrc_train, cv=5)
wrc_rfc_score
```

In [ ]:

```
# cross-validation for the gradient boosting model
wrc_gbc_score = cross_val_score(wrc_gradient_boosting_model, X_wrc_train,
y_wrc_train, cv=5)
wrc_gbc_score
```

Calculate the Mean of the Cross-Validation Scores to Get an Overall Assessment of the Models' Performance

In [ ]:

```
print(f'SVM Model Cross-validation Mean Accuracy: {wrc_svm_score.mean()}')
print(f'Random Forest Model Cross-validation Mean Accuracy:
{wrc_rfc_score.mean()}')
print(f'Gradient Boosting Model Cross-validation Mean Accuracy:
{wrc_gbc_score.mean()}')
```

2. Fill in Mean Values for All Missing Values

In [ ]:

```
# separating target from the rest of the data
X = water_prepared.drop(["remainder__Potability"], axis=1)
y = water_prepared["remainder__Potability"]
```

Using GridSearchCV to Find the Best Value for Parameters for SVC, Random Forest, and Gradient Boosting Algorithms

Split the dataset into 60% training, 20% validation, and 20% testing.

In [ ]:

```
from sklearn.model_selection import train_test_split

X_gs_train, X_validation_test, y_gs_train, y_validation_test =
train_test_split(X, y, test_size=0.4, random_state=42)

X_validation, X_gs_test, y_validation, y_gs_test =
train_test_split(X_validation_test, y_validation_test, test_size=0.5,
random_state=42)

print(X_gs_train.shape, y_gs_train.shape, X_validation.shape,
y_validation.shape, X_gs_test.shape, y_gs_test.shape)
```

GridSearch for SVM

In [ ]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

svm_parameters = {
                    'kernel': ['rbf'],
                    'C': [0.01, 0.1, 1 , 10],
                    'gamma': [0.01, 1, 10]
                  }
```

```
svm = SVC()
svm_gs = GridSearchCV(estimator = svm, param_grid = svm_parameters)
svm_gs.fit(X_gs_train, y_gs_train.values.ravel())

# What is the best estimator from the GridSearch above
svm_winner = svm_gs.best_estimator_

# What is the accuracy of the best estimator
svm_winner.score(X_validation, y_validation)
```

In [ ]:

```
# Display the best estimator
svm_winner
```

GridSearch for Random Forest

In [ ]:

```
from sklearn.ensemble import RandomForestClassifier

rfc_parameters = {
                    'criterion': ['gini', 'entropy'],
                    'max_depth': [2, 4, 6, 8, 10, 13],
                    'random_state':[42]
                  }
rfc = RandomForestClassifier()
rfc_gs = GridSearchCV(estimator = rfc,
                      param_grid = rfc_parameters)
rfc_gs.fit(X_gs_train, y_gs_train.values.ravel())

# What is the best estimator from the GridSearch above
rfc_winner = rfc_gs.best_estimator_

# What is the accuracy of the best estimator
rfc_winner.score(X_validation, y_validation)
```

In [ ]:

```
# Display the best estimator
rfc_winner
```

GridSearch for Gradient Boosting

In [ ]:

```
from sklearn.ensemble import GradientBoostingClassifier

gbc_parameters = {
                    'criterion': ['friedman_mse', 'squared_error'],
                    'max_depth': [2, 4, 6, 8, 10],
```

```
                    'random_state': [42]
                  }
gbc = GradientBoostingClassifier()
gbc_gs = GridSearchCV(estimator = gbc, param_grid = gbc_parameters)
gbc_gs.fit(X_gs_train, y_gs_train.values.ravel())

# What is the best estimator from the GridSearch above
gbc_winner = gbc_gs.best_estimator_

# What is the accuracy of the best estimator
gbc_winner.score(X_validation, y_validation)
```

In [ ]:

```
# Display the best estimator
gbc_winner
```

Split the dataset into a training dataset (80%) and a testing dataset.

In [ ]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

Using Values Retrieved in GridSearch to Train Data

Train a SVM Model (SVC)

In [ ]:

```
# Training SVM Model
svm_model = SVC(kernel='rbf', C=10, gamma=0.01)
svm_model.fit(X_train, y_train.values.ravel())
svm_model
```

Train a Random Forest Model

In [ ]:

```
# Training Random Forest Model
random_forest_model = RandomForestClassifier(max_depth=13,
random_state=42)
random_forest_model.fit(X_train, y_train)
random_forest_model
```

Train a Gradient Boosting Model

In [ ]:

```
# Training Gradient Boosting Model
gradient_boosting_model =
```

```
GradientBoostingClassifier(criterion='squared_error', max_depth=10,
random_state=42)
gradient_boosting_model.fit(X_train, y_train)
gradient_boosting_model
```

Test Models on the X_Test

In [ ]:

```
# Testing the model
svm_y_predict = svm_model.predict(X_test)
random_forest_y_predict = random_forest_model.predict(X_test)
gradient_boosting_y_predict = gradient_boosting_model.predict(X_test)
```

Report the Classification_Report and Accuracy of Each Model on the y_test and y_predict.

In [ ]:

```
from sklearn.metrics import classification_report, accuracy_score

# Evaluate model performances
models = [svm_model, random_forest_model, gradient_boosting_model]
predicts = [svm_y_predict, random_forest_y_predict,
gradient_boosting_y_predict]
model_names = ['SVM', 'Random Forest', 'Gradient Boosting']

for model, predict, name in zip(models, predicts, model_names):
    accuracy = accuracy_score(y_test, predict)
    print(f"------ {name} ------")
    print(f"Accuracy: {accuracy}")
    print("Classification Report:")
    print(classification_report(y_test, predict))
```

Train Models Using KFold Cross-Validation with 5 folds

In [ ]:

```
from sklearn.model_selection import cross_val_score
# cross-validation for svm model
svm_score = cross_val_score(svm_model, X_train, y_train, cv=5)
svm_score
```

In [ ]:

```
# cross-validation for random forest model
rfc_score = cross_val_score(random_forest_model, X_train, y_train, cv=5)
rfc_score
```

In [ ]:

```
# cross-validation for gradient boosting model
gbc_score = cross_val_score(gradient_boosting_model, X_train, y_train,
cv=5)
gbc_score
```

Calculate the Mean of the Cross-Validation Scores to Get an Overall Assessment of the Models' Performance

In [ ]:

```
print(f'SVM Model Cross-validation Mean Accuracy: {svm_score.mean()}')
print(f'Random Forest Model Cross-validation Mean Accuracy:
{rfc_score.mean()}')
print(f'Gradient Boosting Model Cross-validation Mean Accuracy:
{gbc_score.mean()}')
```

5. Graphs for the Best Performing Algorithm

The 2nd way has given the best results, and within that, the random forest model has performed the best.

In [ ]:

```
best_model = random_forest_model
best_model_y_predict = random_forest_y_predict
```

5.1 Confusion Matrix

In [ ]:

```
from sklearn.metrics import ConfusionMatrixDisplay
ConfusionMatrixDisplay.from_predictions(y_test, best_model_y_predict)
```

5.2 ROC-AUC Curve

In [ ]:

```
from sklearn.metrics import RocCurveDisplay
RocCurveDisplay.from_predictions(y_test, best_model_y_predict)
```

5.3 Feature Importances

In [ ]:

```
# referenced from
https://stackoverflow.com/questions/44511636/plot-feature-importance-with-
feature-names

plt.figure(figsize=(8,6))
```

```
feat_importance = pd.Series(best_model.feature_importances_,
index=X.columns)
feat_importance.nlargest(10).plot(kind='barh')
plt.title("Feature Importances - Random Forest")
plt.show()
```

5.4 Calibration Display

In [ ]:

```
from sklearn.calibration import CalibrationDisplay

disp = CalibrationDisplay.from_estimator(best_model, X_test, y_test)
plt.show()
```

Appendix 2 - Project Links

Link to Dataset

Here is the place where we first found our dataset: Water Quality Prediction (7 model) | Kaggle

Here is the original dataset: Water Potability | Kaggle

Link to GitHub

Here is the link to our notebook, for the code related to this project:
https://github.com/Verma-Aakanksha/EECS3401-Project/blob/master/EECS-3401-Project.ipynb

Link to Video

Here is a link to our video, that goes over this project:
https://www.youtube.com/watch?v=BEfTlCsziFo

References

Barla, N. (2023, October 24). *How to do Model Visualization in machine learning?*. neptune.ai. https://neptune.ai/blog/visualization-in-machine-learning

Patel, A. (2021, August 18). *Water quality prediction (7 model)*. Kaggle. https://www.kaggle.com/code/imakash3011/water-quality-prediction-7-model

Plot feature importance with feature names. Stack Overflow. (1963, August 1). https://stackoverflow.com/questions/44511636/plot-feature-importance-with-feature-names