RAJALAKSHMI EDUVERSE

20-02-2025

DATA SCIENCE

PROJECT

Machine Learning - linear

Instructor - Gowthami Shyam

Presented by Keerthana Praveen

Acknowledgement

I would like to express my sincere gratitude to everyone who contributed to the successful completion of this project on Machine Learning - Linear Regression.

Firstly, I extend my heartfelt thanks to my instructor -Mrs. Gowthami shyam, and peers for their valuable guidance, insightful discussions, and continuous support throughout the project. Their expertise has been instrumental in shaping my understanding of data science concepts.

This project has been a great learning experience, helping me apply theoretical knowledge to practical applications in data science.

Thank you.

TABLE
of contents

Introduction

Problem Statement

Access to clean and safe drinking water is a fundamental necessity for human health and well-being. However, due to contamination from industrial waste, agricultural runoff, and poor sanitation, many water sources are unsafe for consumption. The objective of this project is to develop a Machine Learning model that predicts whether a given sample of water is potable (safe to drink) or not based on key water quality parameters such as pH, hardness, dissolved solids, and chemical contaminants.

Motivation

Traditional water testing methods are time-consuming and costly; hence, an ML-based approach can provide a quick and cost-effective solution. By leveraging data-driven techniques, this project aims to assist governments, environmental agencies, and water quality control teams in making informed decisions regarding water safety.

Objectives

1

Data preprocessing involves handling missing values, removing duplicates, and ensuring data consistency for reliable model training. Numerical features are standardized to ensure they lie within a similar range, aiding model performance. Categorical data is encoded appropriately, allowing algorithms to process them effectively. Outliers are detected and treated to avoid bias in the model. The cleaned and processed data is then ready for feature selection and model development.

2

Feature selection identifies the most relevant variables that impact water potability, reducing dimensionality and enhancing model efficiency. Techniques like correlation analysis, mutual information, and recursive feature elimination are employed to evaluate feature importance. Redundant or irrelevant features are removed to prevent noise in the model.

3

Model development involves training various machine learning algorithms, such as Logistic Regression, Decision Trees, and Random Forest, on the preprocessed data. Each model is tuned for optimal performance through hyperparameter optimization. The models are evaluated based on accuracy, precision, recall, and F1-score to find the most suitable one.

4

Model evaluation assesses the predictive performance using metrics like accuracy, precision, recall, and F1-score. A confusion matrix is generated to analyze false positives and negatives. The models are compared to identify the best performer. Cross-validation results are analyzed to check the stability and reliability of the model.

Data Description

Data Sources

The dataset used for this project was obtained from Kaggle, which provides a comprehensive collection of water quality data to predict whether water is potable or not. The dataset is publicly available and contains detailed information on various water quality attributes.

Data Characteristics

The dataset contains a total of 3276 entries and 10 features. The features include both continuous variables, such as pH, Hardness, Solids, Chloramines, and Sulfate, as well as the target variable, Potability, which indicates whether the water is potable (1) or not (0). The dataset consists entirely of numerical values, with 9 float64 columns and 1 int64 column for the target. The features are well distributed, with the mean pH value of 7.07, and the Potability column is imbalanced with a lower mean value (approximately 0.39), suggesting more non- potable than potable water samples. Key statistics show that some features, such as Turbidity, have a wide range of values, with a minimum of 1.45 and a maximum of 6.74, indicating significant variation in the data.

Methodology

- **Phase 1 - Apporach**

The approach for this project follows several key steps aimed at building a model to predict the potability of water. Initially, data preprocessing is performed to clean the data by handling missing values, converting certain features to numeric types, and filling in missing values with the median. Next, feature selection is applied using mutual information to identify the most relevant features for the model. Data is then split into training and testing sets, and SMOTE (Synthetic Minority Over-sampling Technique) is used to balance the dataset. Subsequently, the data is scaled using StandardScaler to normalize the feature values, ensuring the model is trained on uniformly distributed data. Finally, the model is trained using Random Forest Classifier, with hyperparameter tuning applied via GridSearchCV to optimize performance.

- **Phase 2 - Algorithms Used**

The Random Forest Classifier algorithm was selected due to its robustness in handling both classification and regression tasks, ability to work well with large datasets, and its feature importance evaluation capabilities. It was chosen for this project because of its suitability for predicting binary outcomes (potable or non-potable water) and handling imbalanced data with the class_weight='balanced' parameter. Hyperparameter tuning was applied using GridSearchCV to find the optimal settings for the number of trees, maximum depth, and other parameters to improve the model's accuracy. Additionally, SMOTE was used to handle the imbalanced dataset and ensure the model learns effectively from both classes.

Evaluation Metrics

Accuracy

Accuracy: This metric indicates the proportion of correctly classified instances out of the total instances. It helps us understand the overall performance of the model. Before tuning, the accuracy was 63.11%, and after tuning, it improved to 65.09%.

Precision

Precision: Precision measures the proportion of positive predictions that are actually correct. It answers the question: of all the instances classified as "positive" (potable water), how many are actually correct? In this case, the precision for class 0 (non-potable water) before tuning was 0.71, and after tuning, it was 0.72. For class 1 (potable water), precision improved from 0.50 to 0.53 after tuning.

Recall

Recall measures the proportion of actual positives that are correctly identified by the model. It answers the question: of all the actual "positive" instances (potable water), how many did the model correctly identify? For class 0, recall was 0.70 before tuning and 0.73 after tuning, indicating an improvement in detecting non-potable water. For class 1, recall remained relatively stable at 0.51 before and after tuning.

Evaluation Metrics

F1-score

The F1-score is the harmonic mean of precision and recall. It provides a balance between the two, especially when dealing with class imbalances. The F1-score for class 0 was 0.71 before tuning and 0.73 after tuning, indicating improved performance for non-potable water classification. For class 1, the F1-score slightly improved from 0.51 to 0.52.

Confusion Matrix

Confusion Matrix: The confusion matrix provides a detailed breakdown of the model's predictions. It shows:

- True positives (TP): Correct predictions for the positive class (potable water).
- True negatives (TN): Correct predictions for the negative class (non-potable water).
- False positives (FP): Incorrect predictions of the positive class (non-potable water predicted as potable).
- False negatives (FN): Incorrect predictions of the negative class (potable water predicted as non-potable).

Before tuning, the confusion matrix was:
[[290 122]
[120 124]]
After tuning, the confusion matrix was:
[[302 110]
[119 125]]
Overall, the model's performance improved slightly after hyperparameter tuning, with a better balance between precision and recall.

Expected Outcomes

With this project, I anticipate achieving improved predictions in terms of accurately classifying water potability, which is essential for ensuring safe drinking water. By applying various preprocessing techniques, feature selection, and tuning the Random Forest model, the goal is to enhance the model's accuracy and overall performance. Through these efforts, I expect to gain valuable insights into the factors influencing water potability, such as the impact of chemical parameters like pH, sulfate, and turbidity. This project will also help in identifying critical features that most influence the classification of potable vs. non-potable water. Ultimately, I aim to provide a reliable model that can be used in real-world applications for water quality monitoring and decision-making.

Conclusion

In this project, I developed a machine learning model to predict water potability using various features like pH, hardness, and turbidity. After performing extensive data preprocessing, including handling missing values and feature scaling, and applying feature selection techniques, the Random Forest model demonstrated a reasonable level of accuracy. Through model tuning using hyperparameters, we were able to achieve an accuracy of 65%, which marked an improvement from the initial accuracy of 63%. The results indicate that factors such as pH and turbidity significantly impact water potability. Although the model's performance is decent, further improvements could be made by experimenting with additional models, optimizing features, or collecting more data. This project provides valuable insights into the potential of machine learning to assist in water quality assessment, contributing to the broader field of environmental data analysis.

Code

Water potability - Linear Project

[4]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#Load dataset
df = pd.read_csv("water_potability.csv")
df.head(10)
```

[4]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771	0
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0
5	5.584087	188.313324	28748.687739	7.544869	326.678363	280.467916	8.399735	54.917862	2.559708	0
6	10.223862	248.071735	28749.716544	7.513408	393.663396	283.651634	13.789695	84.603556	2.672989	0
7	8.635849	203.361523	13672.091764	4.563009	303.309771	474.607645	12.363817	62.798309	4.401425	0
8	NaN	118.988579	14285.583854	7.804174	268.646941	389.375566	12.706049	53.928846	3.595017	0
9	11.180284	227.231469	25484.508491	9.077200	404.041635	563.885481	17.927806	71.976601	4.370562	0

[24]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               3276 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          3276 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3276 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

[25]:
```
df.describe()
```

[25]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
count	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000	3276.000000
mean	7.074194	196.369496	22014.092526	7.122277	333.608364	426.205111	14.284970	66.407478	3.966786	0.390110
std	1.470040	32.879761	8768.570828	1.583085	36.143851	80.824064	3.308162	15.769958	0.780382	0.487849
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	2.200000	0.738000	1.450000	0.000000
25%	6.277673	176.850538	15666.690297	6.127421	317.094638	365.734414	12.065801	56.647656	3.439711	0.000000
50%	7.036752	196.967627	20927.833607	7.130299	333.073546	421.884968	14.218338	66.622485	3.955028	0.000000
75%	7.870050	216.667456	27332.762127	8.114887	350.385756	481.792304	16.557652	76.666609	4.500320	1.000000
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	28.300000	124.000000	6.739000	1.000000

[5]:
```
#Finding the missing values
df.isnull().sum()
```

[5]:
```
ph                 491
Hardness             0
Solids               0
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

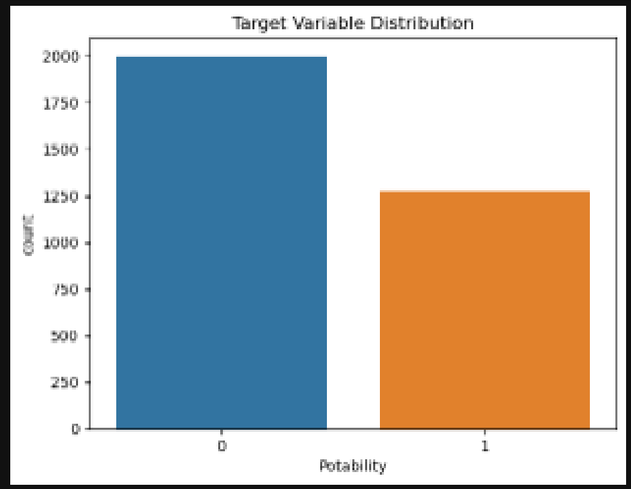
[19]:
```
df.dtypes
```

[19]:
```
ph                 object
Hardness          float64
Solids            float64
Chloramines       float64
Sulfate            object
Conductivity      float64
Organic_carbon    float64
Trihalomethanes    object
Turbidity         float64
Potability          int64
dtype: object
```

Code

```
[21]: df['ph'] = pd.to_numeric(df['ph'], errors='coerce')
      df['Sulfate'] = pd.to_numeric(df['Sulfate'], errors='coerce')
      df['Trihalomethanes'] = pd.to_numeric(df['Trihalomethanes'], errors='coerce')
      df.dtypes

[21]: ph               float64
      Hardness         float64
      Solids           float64
      Chloramines      float64
      Sulfate          float64
      Conductivity     float64
      Organic_carbon   float64
      Trihalomethanes  float64
      Turbidity        float64
      Potability         int64
      dtype: object

[23]: #filling missing values with median
      df['ph'].fillna(df['ph'].median(),inplace = True)
      df['Sulfate'].fillna(df['Sulfate'].median(),inplace = True)
      df['Trihalomethanes'].fillna(df['Trihalomethanes'].median(),inplace = True)
      df.isnull().sum()

[23]: ph               0
      Hardness         0
      Solids           0
      Chloramines      0
      Sulfate          0
      Conductivity     0
      Organic_carbon   0
      Trihalomethanes  0
      Turbidity        0
      Potability       0
      dtype: int64

[7]: #checking for any duplicates
     df.duplicated().sum()

[7]: 0

[8]: #checking for dataset balanced or not
     df['Potability'].value_counts()

[8]: Potability
     0    1998
     1    1278
     Name: count, dtype: int64
```

```
[18]: #visualising the target feature
      import seaborn as sns
      import matplotlib.pyplot as plt

      sns.countplot(x=df['Potability'])
      plt.title("Target Variable Distribution")
      plt.show()
```



Code

[58]:
```
from sklearn.feature_selection import mutual_info_regression
import pandas as pd

# Assuming you have X (features) and y (target) as pandas DataFrames or Series
mi = mutual_info_regression(X, y)
mi_df = pd.DataFrame(mi, index=X.columns, columns=["Mutual Information"])
print(mi_df)
print()
# select features with higher mutual information
print('Selected features for modeling:')
X_selected = X.drop(["Chloramines", "Trihalomethanes", "Solids"], axis=1)
X_selected.head()
```

```
                 Mutual Information
ph                         0.002056
Hardness                   0.026560
Solids                     0.001066
Chloramines                0.000000
Sulfate                    0.000000
Conductivity               0.007077
Organic_carbon             0.004014
Trihalomethanes            0.000000
Turbidity                  0.003075

Selected features for modeling:
```

[58]:

	ph	Hardness	Sulfate	Conductivity	Organic_carbon	Turbidity
0	7.036752	204.890455	368.516441	564.308654	10.379783	2.963135
1	3.716080	129.422921	333.073546	592.885359	15.180013	4.500656
2	8.099124	224.236259	333.073546	418.606213	16.868637	3.055934
3	8.316766	214.373394	356.886136	363.266516	18.436524	4.628771
4	9.092223	181.101509	310.135738	398.410813	11.558279	4.075075

[51]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

# Ensure 'df' is a pandas DataFrame
X = df.drop(columns=['Potability'])  # Drop the target column correctly
y = df['Potability']  # Define the target variable

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from imblearn.over_sampling import SMOTE

# Apply SMOTE to the training set for balanced dataset
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)

# Check the class distribution after SMOTE
print("Class distribution after SMOTE:")
print(y_resampled.value_counts())
```

```
Class distribution after SMOTE:
Potability
0    1586
1    1586
Name: count, dtype: int64
```

Code

```
[52]: #scaling dataset
      scaler = StandardScaler()
      X_resampled_scaled = scaler.fit_transform(X_resampled)  # Fit on training data
      X_test_scaled = scaler.transform(X_test)  # Transform test data
```

```
[57]: from sklearn.ensemble import RandomForestClassifier

      # Instantiate the Random Forest model
      rf_model = RandomForestClassifier(random_state=42, class_weight='balanced')

      # Train the model on the scaled training data
      rf_model.fit(X_resampled_scaled, y_resampled)

      # Make predictions on the scaled test data
      y_pred_rf = rf_model.predict(X_test_scaled)

      # Evaluate the model
      accuracy_rf = accuracy_score(y_test, y_pred_rf)
      print(f"Random Forest Accuracy: {accuracy_rf}")
      print("Classification Report:\n", classification_report(y_test, y_pred_rf))
      print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.6310975609756098
Classification Report:
              precision    recall  f1-score   support

           0       0.71      0.70      0.71       412
           1       0.50      0.51      0.51       244

    accuracy                           0.63       656
   macro avg       0.61      0.61      0.61       656
weighted avg       0.63      0.63      0.63       656

Confusion Matrix:
 [[290 122]
 [120 124]]
```

```
[60]: # Ensemble model algorithm
      import lightgbm as lgb
      from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

      # Define the model
      lgb_model = lgb.LGBMClassifier(random_state=42)

      # Hyperparameter grid for LightGBM
      param_grid_lgb = {
          'n_estimators': [100, 200, 300],
          'max_depth': [10, 20, 30],
          'learning_rate': [0.01, 0.05, 0.1],
          'subsample': [0.8, 0.9, 1.0]
      }

      # Setup GridSearchCV for LightGBM
      grid_search_lgb = GridSearchCV(estimator=lgb_model, param_grid=param_grid_lgb,
                                     cv=5, n_jobs=-1, verbose=2, scoring='accuracy')

      # Fit the model
      grid_search_lgb.fit(X_resampled_scaled, y_resampled)

      # Get the best parameters and model
      best_params_lgb = grid_search_lgb.best_params_
      best_lgb_model = grid_search_lgb.best_estimator_

      # Predict with the best model
      y_pred_best_lgb = best_lgb_model.predict(X_test_scaled)

      # Evaluate the best model
      accuracy_best_lgb = accuracy_score(y_test, y_pred_best_lgb)
      print(f"Best LightGBM Accuracy: {accuracy_best_lgb}")
      print("Classification Report:\n", classification_report(y_test, y_pred_best_lgb))
      print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred_best_lgb))
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
[LightGBM] [Info] Number of positive: 1586, number of negative: 1586
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000696 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 2295
[LightGBM] [Info] Number of data points in the train set: 3172, number of used features: 9
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500000 -> initscore=0.000000
Best LightGBM Accuracy: 0.6387195121951219
Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.70      0.71       412
           1       0.51      0.54      0.53       244

    accuracy                           0.64       656
   macro avg       0.62      0.62      0.62       656
weighted avg       0.64      0.64      0.64       656

Confusion Matrix:
 [[287 125]
 [112 132]]
```

In conclusion, we use the Random Forest model with hyperparameter tuning, as it provides the best score here.

References

Raja Lakshmi Eduverse Data Science Course – Mrs. Gowthami Shyam, Instructor

- Course: Data Science
- Institution: Rajalakshmi Eduverse
-
- This course provided foundational knowledge and practical skills in data science, including data preprocessing, machine learning algorithms, and model evaluation.

Kaggle (2025). "Water Potability Dataset."

Retrieved from https://www.kaggle.com/datasets

The dataset used for this project was sourced from Kaggle, which provided relevant water quality data for analysis and model development.


Thank You