Applied Machine Learning Project

Water Quality

Ahsan Zahoor Khan (x6087870@student.uwasa.fi)

Towkir Ahmed (x1001005@student.uwasa.fi)

Eemil Aalto-Setälä (x7923898@student.uwasa.fi)

Vaasa 2023

Project Contribution

In our collaborative project, each of the three team members contributed equally to its success. We began by collectively planning and strategizing, ensuring that every member's voice was heard. Work distribution was fair, with each member taking on specific responsibilities. Our diverse skills were utilized, and knowledge sharing was a priority to foster growth within the team. Flexibility and adaptability were key as we faced challenges, and a robust peer-review process ensured the quality of our work. The success of the project is a result of our combined effort and commitment.

VAASAN YLIOPISTO

Author: Ahsan Zahoor Khan, Towkir Ahmed, Eemil Aalto-Setälä
Project title: Water Quality
Supervisor: Petri Välisuo
Number of pages: 31

ABSTRACT:
Potable Water...
Potable water, also known as drinking water, is water that is safe for ingestion, either when drunk directly or consumed indirectly through food preparation. The main source of our drinking water is the fresh-water bodies. Our water sources get contaminated in various ways. These contaminations affect the different chemical parameters that are present in the water bodies. The potability of the water can be greatly impacted by measurements that are either too high or too low for certain chemical parameters present in water bodies. To determine if drinking water is safe, it should be tested for pH, ions (sodium, chloride, iron, manganese, fluoride, sulfate), total dissolved solids, and hardness etc. In this project, we used machine learning techniques to develop an autonomous system that can determine, based on a variety of water parameters, whether or not a given supply of water is potable. After thorough processing and implementation, we were able to achieve satisfactory performance from our applied machine learning algorithms.

Keywords: Machine Learning, Water Quality, Potability, Random Forest, SVM, PCA

Contents

1 Introduction

1.1 Objective

The main objective of this project in the course is 'Applied Machine Learning' is to utilize machine learning methodologies to accurately predict the potability/quality of water. This involves analyzing a dataset containing various water quality metrics and implementing a predictive model that classifies water samples as either potable or non-potable. The project aims to demonstrate the effective application of machine learning techniques in environmental data analysis, particularly in this case assessing water quality.

Access to clean and safe drinking water is a critical public health issue. In many regions, assessing the potability of water is a challenging task full of complexities due to the varying chemical compositions and potential contaminants. The significance of this project problem lies in its potential to in theory provide a fast, reliable, and scalable solution for water quality assessment, which can be particularly beneficial in settings where resources are limited. By automating the process of water quality assessment, the successful application of such models could help significantly in health and safety measures related to water consumption.

The challenge of this project, like in most machine learning projects, lies in finding an effective way to process and analyze the dataset, which includes a range of water quality parameters and comes with some issues such as missing values. The task involves the application of machine learning models and preliminary steps like data preprocessing and feature scaling, which are important for the accuracy and reliability of the models' predictions.

1.2 Overview

The project begins with an exploratory data analysis, followed by handling missing values in the dataset through mean imputation. Feature scaling is conducted to normalize the data, and PCA is applied for dimensionality reduction. The analysis then moves to the application and comparison of various machine learning models, including Logistic Regression, Random Forest, and SVM. Each model's performance is evaluated and compared to determine the most effective approach for predicting water potability, with a focus on achieving high accuracy and as reliable classification results as possible.

The scope of this project is focused on demonstrating practical skills in applying machine learning techniques in an academic context. The project serves more as an exercise in data analysis and model development, rather than a comprehensive research endeavor. The findings and methodologies presented are intended to showcase the ability to navigate and implement machine learning tools effectively, rather than providing detailed insights into the field of water quality assessment.

2 Dataset

Our group had initially chosen food waste analysis data for the project. The dataset, however, was deemed to not have enough non-null values to make meaningful and accurate predictions. After dropping the food waste dataset, a water potability dataset, which was uploaded to Kaggle by user Aditya Kadiwal, was chosen to be used as the foundation of this project.

2.1 Dataset Details

The water potability dataset includes water quality metrics for 3,276 different water bodies. It comprises the following features:

- **pH**: Containing 2785 non-null values, pH is a measure of the acid-base balance of water
- **Hardness**: Available for all 3276 samples in the dataset, indicating the presence of calcium and magnesium salts
- **Solids (Total dissolved solids – TDS)**: Present in all samples, reflecting the water's mineral content
- **Chloramines**: Also available for all samples, showing the level of chloramines used as disinfectants
- **Sulphate**: Contains 2495 non-null values, a measure of sulfate concentration
- **Conductivity**: Complete data for all samples, indicating water's ionic concentration
- **Organic Carbon**: Available for all samples, representing the total organic carbon content
- **Trihalomethanes**: With 3114 non-null values, these are byproducts of chlorination

- **Turbidity**: Data available for all samples, measuring water's cloudiness
- **Potability**: The target variable, indicating if water is potable (safe to drink), with 1 representing potable and 0 non-potable

Each feature in the dataset is relevant in assessing water quality. Below the feature relevance of each feature is further explained:

- **pH**: The pH level of water is crucial because extreme pH levels (too acidic or too alkaline) can lead to harmful chemical reactions that can be dangerous to human health. Water with balanced pH levels is less corrosive and more suitable for consumption
- **Hardness**: Excessively hard water can damage pipes and appliances due to scale build-up. It can also lower the effectiveness of soap and detergent in cleaning
- **Solids (Total dissolved solids – TDS)**: High TDS levels can lead to undesirable taste and color, and in extreme cases can carry harmful contaminants that pose health risks
- **Chloramines**: Chloramine levels in water are important as they need to be sufficient to kill harmful microorganisms, but not so high as to pose a health risk themselves
- **Sulphate**: High sulfate levels can lead to a bitter taste, and at very high levels they can have a laxative effect or other health impacts
- **Conductivity**: A measure of water's ability to conduct electricity, which correlates with the amount of dissolved salts or ions present. High conductivity can indicate high concentration of potentially harmful substances, making it a valuable indicator of water quality
- **Organic Carbon**: High levels can suggest contamination from decaying natural matter or human activity. Organic carbon can affect water taste and color and can indicate potential for harmful chemical reactions in the water

- **Trihalomethanes**: High levels are concerning as they are linked to health issues, including an increased risk of cancer
- **Turbidity**: High turbidity can shield harmful pathogens from disinfection process and can also indicate runoff or erosion issues, affecting water quality
- **Potability**: Safe drinking water is essential for health, and accurately predicting water potability is important public health

2.2 Exploratory Data Analysis

Our dataset contains 3276 entries and 10 features. Figure 1 shows the distribution of data and number of data for each features.

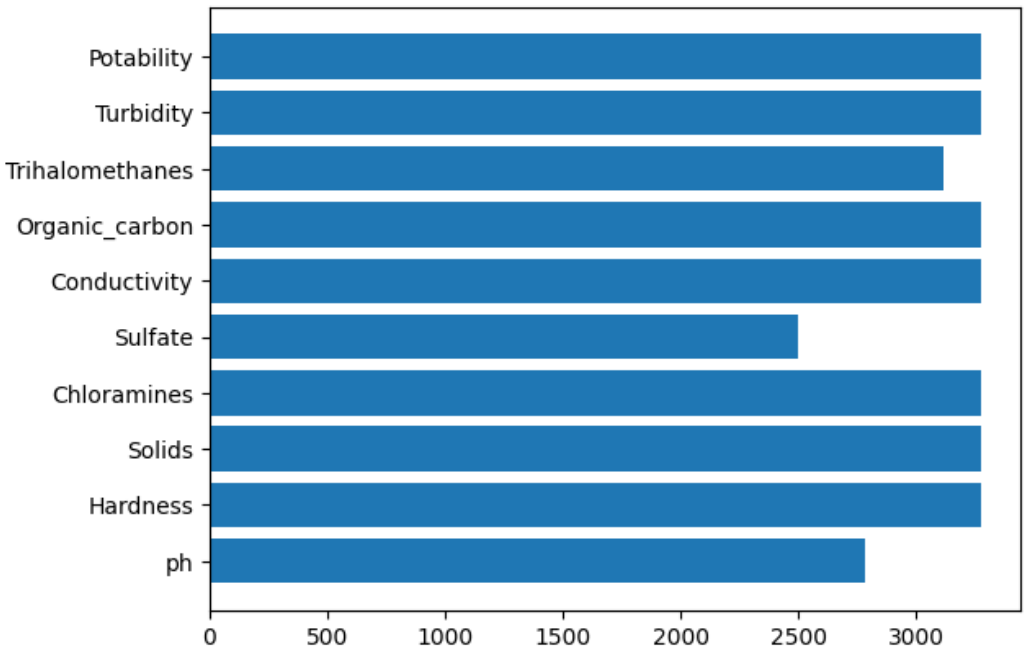

Figure 1. Data Distribution.

It was mentioned earlier that our dataset contains some missing values. It can also be seen from Figure 1. To get the total number of missing values for each feature, we generate the following figure.

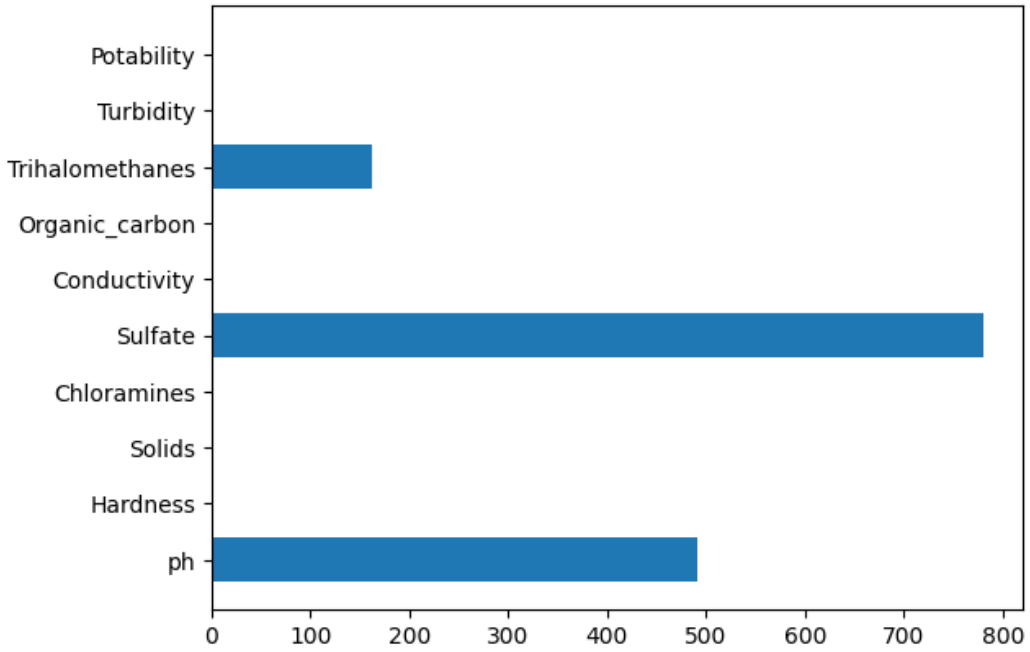

Figure 2. Barplot of Missing Values.

From Figure 2 we can see that, only pH, Triathlomethanes and Sulfate has missing values. The code for generating the plots can be found in 7.

3 Data Preprocessing

Data preprocessing is critical in machine learning and data analysis because it converts raw data into a more understandable and efficient format, thereby improving data quality and model accuracy. This process is vital because real-world data often contains inconsistencies, missing values, and various scales, which can significantly impact the performance of machine learning models. Effective preprocessing improves data quality and makes the resulting models more accurate and reliable. It can include handling missing values, feature scaling, normalization, and sometimes more complex transformations like feature engineering. These steps help in reducing noise, handling anomalies, and enhancing the overall predictive capability of the machine learning models Bowne-Anderson (2016).

As mentioned earlier, our dataset contains 3276 rows and 10 columns. Our target class is 'Potability' which has only 0 or 1 values. 0 means the water is not Potable and 1 indicates it is Potable. Additionally, for each sample, a total of 9 features are considered. To help understand the relationship of the features, we have generated a pairplot. Figure 3 shows the pairplot.

For our data pre-processing step, we followed the steps mentioned below.

3.1 Handling Missing Values

Handline missing values is one of the most important parts of a machine learning project. The way the missing data are handled can have a great impact on the performance of the machine learning models. In our dataset, all columns had some values missing. To handle that, we followed two approaches:

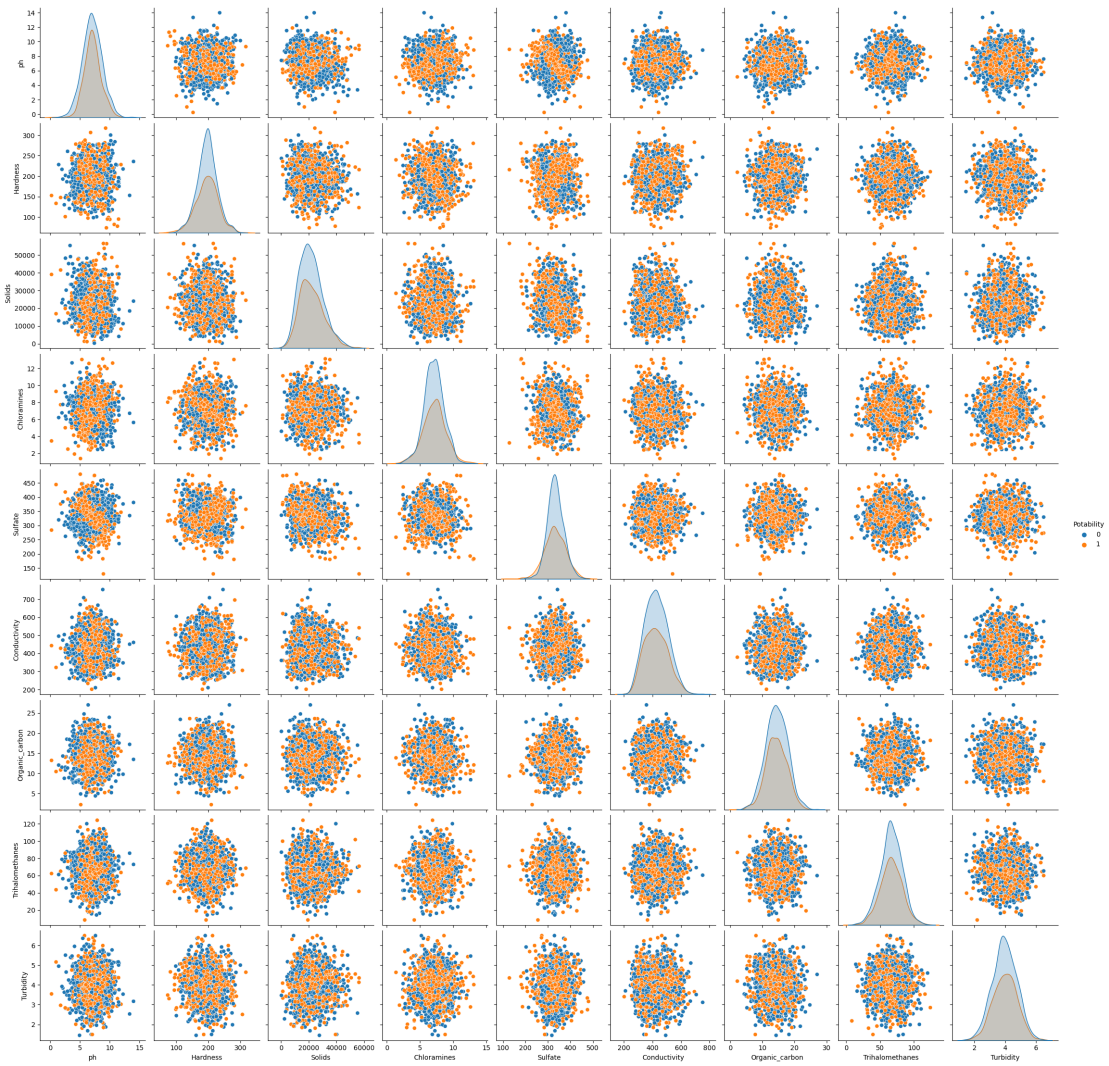

Figure 3. Pairplot among features.

3.1.1 Dropping the missing values

Dropping the missing values is one of the ways to handle missing data. It aids in reducing the dataset's size and complexity, avoiding errors or biases that can arise from imputing or replacing missing values, and preserving the data's original distribution and variance. However, it may result in the loss of valuable information and a reduction in sample size, which may affect the validity and generalizability of the analysis. After dropping all values from the columns, we get only 2011 rows. In this case, we got all the data has do not have any missing values, but the tread off is, we lost a significant amount of data.

3.1.2 Taking mean for the missing values

Taking mean from each feature for the missing values is another way to handle missing data. Even though the data inserted may not be exact, this method helps to keep all the data which is very important to achieve a good performance.

We tried both the approaches and the end result can be found in the result analysis section.

3.2 Scaling

Scaling is important in machine learning because it brings all features to the same magnitude, preventing variables with larger scales from dominating the model and ensuring that the model is not affected by feature scale. It also improves the performance of machine learning algorithms that calculate distances between data points, which can be critical for the convergence of algorithms such as non-penalized logistic regression and for creating different model fits than the fit with un-scaled data. To scale our data, we used the Standard Scaler provided by scikit learn library. This method was particularly suitable for the Water Potability dataset, because it comprises features with varying ranges and units of measurement. StandardScaler normalizes these features by removing the mean and scaling them to unit variance. This ensures that each feature contributes equally to the model's performance and is not biased by the scale of its values. This step is especially important for algorithms like SVM, which are sensitive to the magnitude of input features. For example, in this dataset, features like TDS (Total Dissolved Solids) and hardness vary greatly in scale, and without normalization, there's a risk that larger-scaled features could disproportionately influence the model's learning process. Standardizing these features ensures a more balanced and effective model training. Khoong, W. H. (2021). The code for scaling can be found in Appendix 8.

3.3 Dimensionality Reduction

Dimentionality Reduction is used to remove features that have low impact on the target class which makes classifiers more efficient because it eliminates the need for some unnecessary calculations. Dimensionality reduction also reduces the risk of overfitting in machine learning models. Li, L. (2019). Principal Component Analysis is a popular technique for reducing dimension. It reduces the number of features in a dataset while retaining as much information as possible. We have also implemented dimensionality reduction in our dataset. We used the scaled data and got the following pairplot.

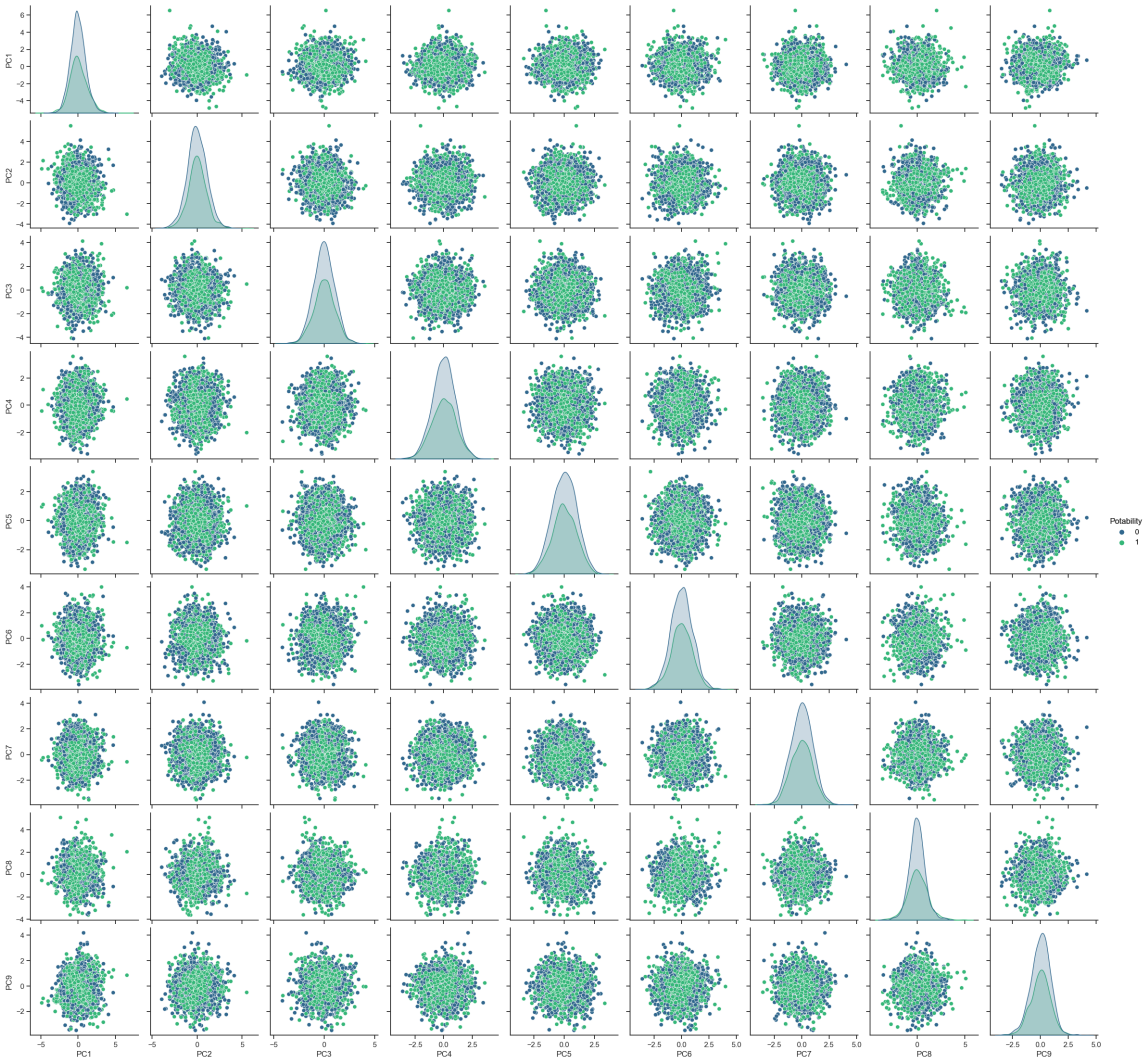

Figure 4. Pairplot among features after using PCA.

From the code shown in 1, we get the variance ratios for each features are 0.13338306, 0.12632158, 0.11732755, 0.11370298, 0.11044415, 0.10744622, 0.10566013, 0.0985114 and 0.0872029. From the code and output of PCA variance ratio (Appendix 1) and the pairplot (Figure 4), we can see that, all our features have similar variance and there are not much difference in the pairplots before and after doing PCA. Thus, we can not remove any feature from our dataset as all of them are significant.

3.4 Data Splitting

Since we have to keep some of the data for testing our machine learning models, it is best to use data splitting. The balance is key in model evaluation, as it allows for both robust learning and an unbiased assessment of the model's generalization capability. For this, we train our models with some data and rest of the data are kept for testing our models to see how our models perform when it is given some unseen data. We use the train test split method provided by scikit learn for this. For our case, we kept 33% data for testing and used the random state 42 (Appendix 2).

4 Machine Learning Models

The process of model selection is pivotal in any machine learning task as it lays the groundwork for the entire predictive analysis. The chosen models for this study are Logistic Regression, Random Forest, and Support Vector Machines (SVM), each selected for their unique strengths and compatibility with the characteristics of the dataset. Since our main task is classification, we have used some machine learning classifiers provided by scikit learn library. At first, we have used all the default parameters and then we used Grid Search CV to find the best possible hyperparameters from our best performing models. The models are listed below:

4.1 Support Vector Machine

Support Vector Machines (SVMs) are important in machine learning due to their ability to handle high-dimensional data, perform well with small datasets, and model non-linear decision boundaries, making them suitable for a wide range of applications. SVM uses kernels which is a mathematical function used to transform data into a higher-dimensional space, enabling the algorithm to find a linear decision boundary in non-linear problems. SVM focuses on maximizing the margin between classes, ensuring a clear and optimized boundary for classification, crucial for datasets with potential overlaps between classes (Raz, A. (2022)). Kernels are essential for handling complex datasets and non-linear relationships between features, making SVMs suitable for a wide range of applications. SVM has various kernels. Among them, we have used Linear Kernel and Radial Basis Function (RBF) Kernel. Hyperparameters: C=1.0, kernel='rbf' and 'linear', degree=3, gamma='scale'.

4.1.1 SVM-Linear

Linear kernel of SVM is known for its ability to handle high-dimensional data, to work well with small datasets, and to model linear decision boundaries. We have used SVM linear

in our project.

4.1.2 SVM-RBF

RBF kernel can handle datasets with non linear relationships as well as high dimensional data which makes it a very efficient classifier. It works by mapping the input data into a higher-dimensional feature space, where the classes can be separated by a hyperplane. We took interest in RBF kernel as it can perform very well in complex dataset.

4.2 Random Forest

Random Forest was selected for its robustness in handling different kinds of datasets. It is capable of capturing complex, non-linear interactions between features, useful for an accurate analysis of our water quality dataset, which includes various metrics. This model is known for its reduced risk of overfitting, thanks to its ensemble nature (Sruthi, E. (2023)). The importance of Random Forest lies in its ability to reduce overfitting in decision trees, improve accuracy, handle high-dimensional data, and provide feature importance, making it suitable for both classification and regression tasks. Notable hyperparameters: n_estimators=100, max_depth=None, min_samples_split=2, min_samples_leaf=1 .

4.3 Logistic Regression

As a widely used and easily interpretable model, Logistic Regression provides a solid performance in classification taks. Because of its simplicity, efficiency, and ability to provide a natural probabilistic view of class predictions, logistic regression is important (Kleimann, S (2020)). It is commonly used for classification tasks, can handle both binary and multiclass problems, and is simpler to implement and interpret than other machine learning algorithms. Notable hyperparameters: penalty='l2', C=1.0 .

4.4 Gradient Boosting Classifier

Gradient boosting is a machine learning technique for both regression and classification problems. Gradient boosting is well-known for its ability to handle complex datasets and produce cutting-edge results in a variety of prediction tasks; it can also provide estimates of feature importance from a trained predictive model. It is an ensemble method. Hyperparameters used for this classifier are: loss='log_loss', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1 (Aliyev, V. (2020)).

4.5 Bagging Classifier

The Bagging Classifier is an ensemble learning technique that improves predictive performance and model robustness by training multiple models on different subsets of the data and combining their predictions. This approach helps to reduce overfitting and increase the accuracy of the model, making it suitable for a wide range of applications. Hyperparameters used for this classifier are: estimator=SVM, n_estimators=10, *, max_samples=1.0, max_features=1.0 (Elabd, M. (2022)).

5 Results

To analyse our results, at first we present the performances of our machine learning algorithms in tables. Then we tune the hyperparameters for our best performing algorithms. Lastly, we show the cross validation results.

5.1 Performance Analysis

As mentioned earlier, we tried to handle our missing data in two ways, by dropping and by taking the mean. Table 1 shows the performance of various implemented algorithms.

Table 1. Performance of algorithms after dropping the missing values.

Algorithm	Accuracy	F1-Score	Precision	Recall
SVM-Linear	60.09	30.05	50.00	37.54
SVM-RBF	68.52	68.32	63.73	63.66
Random Forest	67.17	66.05	62.80	62.78
Logistic Regression	60.39	57.57	51.01	41.55
Bagging Classifier with SVM	67.62	66.88	62.98	62.89
Gradient Boosting Classifier	64.16	62.14	61.24	61.38

For our second approach by taking the mean, we get the following results shown in Table 2.

Table 2. Performance of algorithms after substituting the missing values by their mean.

Algorithm	Accuracy	Precision	Recall	F1-Score
SVM-Linear	61.19	35.25	50.82	36.88
SVM-RBF	70.67	72.10	63.55	63.55
Random Forest	69.30	67.80	62.13	62.33
Logistic Regression	63.00	31.68	50.10	39.25
Bagging Classifier with SVM	69.55	68.33	65.38	65.89
Gradient Boosting Classifier	65.27	66.25	65.36	65.49

From both table 1 and table 2, we can see that the results are slightly different and the algorithm performs slightly better when using the mean values. This may be because in

this case, the models do not lose significant amount of data. In addition, we see that, the best performing models are SVM-RBF and Random Forest. Bagging classifier also performs well as it used SVM as estimator but our SVM-RBF model outperforms all other models.

5.2 Getting The Best Hyperparameters

It can be seen from the tables that, our best performing models are SVM-RBF and Random Forest. We attempted to determine the optimal hyperparameters by using the Grid Search CV method from Scikit-Learn library.

5.2.1 Hyperparameter tuning for SVM-RBF

The main hyperparameters for SVM-RBF are Regularization Parameter 'C' and Kernel Coefficient 'gamma'. We tried several values for both hyperparameters. The values we tried for 'C' are 0.1, 1, 10, 100 and 1000. On the other hand, the values tried for 'gamma' are 1, 0.1, 0.01, 0.001 and 0.0001. The code for this can be found in Appendix 3. The best values for the hyperparameters we have found are: C=1, gamma=0.1.

5.2.2 Hyperparameter tuning for Random Forest

The main hyperparameters for Random Forest are number of trees 'n_estimators', the maximum depths of the trees 'max_depth', the minimum number of samples required to split an internal node 'min_samples_split' and the minimum number of samples required to be at a leaf node 'min_samples_leaf'. We tried several values for all the hyperparameters. The values we tried for 'n_estimators': [50, 100, 200], for 'max_depth': [None, 10, 20, 30], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4]. The code for this can be found in Appendix 4. The best values for the hyperparameters we have found are: 'max_depth': None, 'min_samples_leaf': 2, 'min_samples_split': 10, 'n_estimators': 100.

5.2.3 Result Analysis of Grid Search CV

The performance metrics after Grid Search is shown in table 3.

Table 3. Performance of algorithms after dropping the missing values.

Algorithm	Accuracy	F1-Score	Precision	Recall
SVM-RBF	71.67	72.13	68.54	68.33
Random Forest	70.88	70.67	65.22	65.31

From the above table, we can see that, grid search makes our algorithms slightly better. Even though the overall performance are not very much improved, the recall and f1-score have improved significantly.

5.3 Cross Validation

We also tried cross validation to improve the models further. We tried CV=10 for both SVM-RBF and Random Forest. The codes can be found in Appendix 5 and 6. The cross validation results show significant improvement. For SVM-RBF, we were able to achieve the accuracy up to 75.24% and for Random Forest up to 73.33%.

6 Conclusion

Water Potability is a very important factor. We tried to implement some machine learning models to classify water sources Potability based on different features or factors. Even though the performance of our models were not very high, it was satisfactory considering the fact that we had lots of missing values. Also, a larger dataset would provide better performances from our models. We aim to improve our models further, implement more algorithms and if possible, enrich the dataset with more data.

Bibliography

Aliyev, V. (2020). *Gradient boosting classification explained through python. towards data science*. Retrieved from https://towardsdatascience.com/gradient-boosting-classification-explained-through-python-60cc980eeb3d

Bowne-Anderson. (2016). *Preprocessing in data science (part 1): Centering, scaling and knn*. Retrieved from https://www.datacamp.com/tutorial/preprocessing-in-data-science-part-1-centering-scaling-and-knn

Elabd, M. (2022). *What is bagging classifier?. medium*. Retrieved from https://medium.com/@arch.mo2men/what-is-bagging-classifier-45df6ce9e2a1

Khoong, W. H. (2021). *Why scaling your data is important. medium*. Retrieved from https://medium.com/codex/why-scaling-your-data-is-important-1aff95ca97a2

Kleimann, S. (2020). *Logistic regression for binary classification. towards data science*. Retrieved from https://towardsdatascience.com/logistic-regression-for-binary-classification-56a2402e62e6

Li, L. (2019). *Principal component analysis for dimensionality reduction. towards data science*. Retrieved from https://towardsdatascience.com/principal-component-analysis-for-dimensionality-reduction-115a3d157bad

Raz, A. (2022). *Everything about support vector classification — above and beyond. towards data science*. Retrieved from https://towardsdatascience.com/everything-about-svm-classification-above-and-beyond-cc665bfd993e

Sruthi, E. (2023). *Understanding random forest algorithms with examples. analytics vidhya*. Retrieved from https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/

Appendices

Appendix 1. Python code to get PCA values for each feature

```
pca = PCA()
pca_result = pca.fit_transform(A_pairplot_standardized)
varexp= pca.explained_variance_ratio_
print (varexp)

[0.13338306 0.12632158 0.11732755 0.11370298
0.11044415 0.10744622 0.10566013 0.0985114
0.08720294]
```

Listing 1. PCA Code.

Appendix 2. Train test split code

```
X_train, X_test, y_train, y_test =
train_test_split(X_scaled, Y, test_size=0.33,
random_state=42)
```

Listing 2. PCA Code.

Appendix 3. Grid Search CV for SVM-RBF

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

param_grid = {'C': [0.1, 1, 10, 100, 1000],
      'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
      'kernel': ['rbf']}

grid = GridSearchCV(SVC(), param_grid, refit = True,
verbose = 3)
grid.fit(X_train, y_train)
print("Best Parameters:", grid.best_params_)
best_model = grid.best_estimator_
y_pred = best_model.predict(X_test)
```

Listing 3. Grid Search CV Code for SVM-RBF.

Appendix 4. Grid Search CV for Random Forest

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

param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
clf = RandomForestClassifier(random_state=42)
grid = GridSearchCV(random_forest_model,
param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)
print("Best Parameters:", grid.best_params_)
best_model = grid.best_estimator_
y_pred = best_model.predict(X_test)
```

Listing 4. Grid Search CV Code for Random Forest.

Appendix 5. Cross Validation Code For SVM-RBF

```
from sklearn.model_selection import cross_val_score
classifier = SVC(kernel = 'rbf', C=1, gamma=0.1,
random_state = 42)
cv_score= max(cross_val_score(classifier, X_scaled, Y, cv=10))
print(cv_score)
```

Listing 5. Cross Validation for SVM Code.

Appendix 6. Cross Validation Code For Random Forest

```
from sklearn.model_selection import cross_val_score
classifier = RandomForestClassifier('max_depth': None,
'min_samples_leaf': 2, 'min_samples_split': 10,
'n_estimators': 100, random_state=42)
cv_score= max(cross_val_score(classifier, X_scaled, Y, cv=10))
print(cv_score)
```

Listing 6. Cross Validation for Random Forest Code.

Appendix 7. Code for Generating Bar Plots

```
# Getting the bar plot for all features
plt.barh(original_df.count().index, original_df.count())

# Getting the bar plot for missing values
nan_all = original_df.isna().sum()
plt.barh(nan_all.index, nan_all.values)
```

Listing 7. Codes to Generate Bar Plots.

Appendix 8. Code for scaling the data

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_PCA = pd.DataFrame(scaler.fit_transform(X_scaled), columns=df
```

Listing 8. Codes to scale the data.