National College of Ireland

Configuration Manual: Groundwater Quality Predictive Analysis Using Machine Learning Techniques: Ireland

MSc Research Project
MSc in Data Analytics

Leslie Rebeca Monroy Ochoa
Student ID: x23169761

School of Computing
National College of Ireland

Supervisor: Jaswinder Singh

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Leslie Rebeca Monroy Ochoa
Student ID:	x23169761
Programme:	MSc in Data Analytics **Year:** 2023-2024
Module:	Research Project
Supervisor:	Jaswinder Singh
Submission Due Date:	Monday 16th September 2024
Project Title:	Groundwater Quality Predictive Analysis Using Machine Learning Techniques: Ireland
Word Count:	1269 words **Page Count:** 9 pages

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Leslie Rebeca Monroy Ochoa

Date: 13th September 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual: Groundwater Quality Predictive Analysis Using Machine Learning Techniques: Ireland

Leslie Rebeca Monroy Ochoa
x23169761

1 Introduction

This configuration manual provides a guide to the environmental setup required for the research project titled: Groundwater Quality Predictive Analysis Using Machine Learning Techniques: Ireland. The objective of this document is to present the configuration used during all the stages of the research that allowed to answer the research question and meet the research objectives.

1.1 Research Project Objectives

The main objective of the research project was to implement and evaluate four supervised learning machine models to analyse and predict groundwater quality parameters in Ireland. The selected models were Decision Tree, Random Forest, Extreme Gradient Boosting (XGBoost), and Support Vector Machine (SVM). The evaluation consisted of utilising Accuracy, Precision, Recall, and F1 score as performance metrics to verify the model effectiveness.

2 Configuration Setup

This section shows the hardware and software utilisation for the research project.

2.1 Hardware

Hardware	**Specification**
Processor	Intel(R) Core(TM) i5-10210U CPU @ 1.60GHz 2.11 GHz
RAM	8 GB
System	Operating system-64 Bit, x64 processor
Operating System	Windows 11

Table 1. Hardware

2.2 Software

R Studio

Software	Specification
Version	2023.06.2
Build	561
Release	"Mountain Hydrangea" Release (de44a3118f7963972e24a78b7a1ad48b4be8a217, 2023-08-25) for windows Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) RStudio/2023.06.2+561 Chrome/110.0.5481.208 Electron/23.3.0 Safari/537.36
Library	**Version**
readxl	4.3.3
dplyr	4.3.3
ggplot2	4.3.3
reshape2	4.3.3
tidyr	4.3.2
gridExtra	4.3.2
lubridate	4.3.2
ggcorrplot	4.3.3
randomForest	4.3.3
e1071	4.3.3

Table 2. R Studio Specifications

Anaconda - Jupyter Notebook

Software	Version
Anaconda Navigator	2.4.2
Jupyter Notebook	6.5.4

Table 3. Jupyter Notebook Specifications

3 Data Collection

The dataset was obtained from the Environmental Protection Agency Geo Portal (https://gis.epa.ie/). The dataset, which has 16,231 rows and 304 columns, provides water quality metrics for the groundwater stations in Ireland that were monitored between 1990 and 2022. Important metrics like pH, temperature, dissolved oxygen, conductivity, and coliform bacteria are included in the dataset, along with identification information like site name, county, and sample date.

The dataset can be downloaded following this link https://gis.epa.ie/GetData/Download →Water Quality and Monitoring → Groundwater Quality (Excel) 1990 – 2022. Where you are required to enter and validate your email address to get the download link.

4 Data Preparation

Most of the data preparation was performed in R Studio using R as the programming language.

The first step for this phase was to install and load the required libraries to prepare, clean and analyse the data as shown in Figure 1.

```
#Installing the necessary libraries
#install.packages("readxl")
#install.packages("dplyr")
#install.packages("ggplot2")
#install.packages("reshape2")
#install.packages("tidyr")
#install.packages("gridExtra")
#install.packages("lubridate")
#install.packages("ggcorrplot")
#install.packages("randomForest")
#install.packages("e1071")

#Loading the required libraries
library(readxl)
library(dplyr)
library(ggplot2)
library(reshape2)
library(tidyr)
library(gridExtra)
library(lubridate)
library(ggcorrplot)
library(randomForest)
library(e1071)
```

Figure 1. Required Libraries

Next, the dataset was imported and loaded into the environment, showing the structure and summary of the features to start the data analysis, shown in Figure 2.

```
#Importing and reading groundwater quality dataset
file = 'D:/Documentos Personales/MSC in Data Analytics/Research Project/Dataset/MON_GW

dataset = read_excel(file)

#Groundwater Quality Dataset
head(dataset)

summary(dataset)

colnames(dataset)
```

Figure 2. Groundwater Quality Dataset Load

Following, the data was cleaned and transformed, the columns with more than 10% of NA or “--” were deleted, the unit description row was also removed as shown in Figure 3. To efficiently handle the data, the "--" were replaced with "NA" and special characters like <> were also removed, leaving only numeric values.

```
#Deleting Unit row of the dataset
dataset = dataset[-1, ]

#Replacing "-" and "--" with NA
dataset[dataset == "-"] = NA
dataset[dataset == "--"] = NA
```

Figure 3. Data Cleaning

Once these steps were executed, the feature distribution was then validated, and skewness was checked. Subsequently, a function was executed to impute mean or median according to skewness, presented in Figure 4 below.

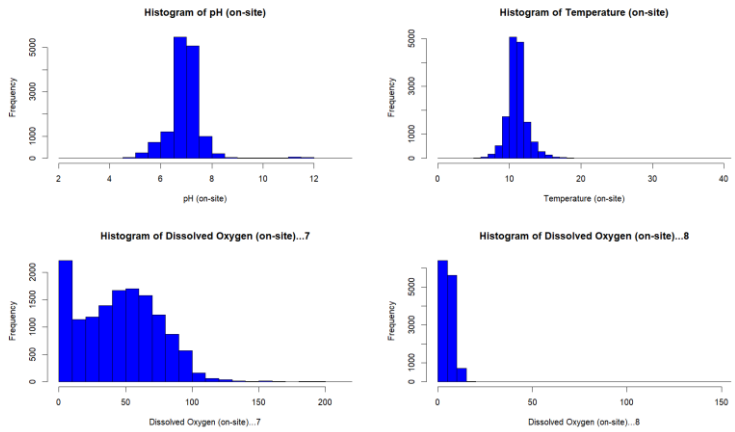

Figure 4. Histograms - Feature Distribution

Next the Sample Date was converted to a date format as shown in Figure 5.

```
#Convert Sample Date to date format
#Checking dataframe format
head(data$`Sample Date`)
data$`Sample Date` = as.numeric(data$`Sample Date`)
data$`Sample Date` = as.Date(data$`Sample Date`, origin = "1899-12-30")
```

Figure 5. Sample Date Conversion

To finish with the preparation and analysis of the data, several types of visualizations were generated such as correlation matrix, heatmap, time series, and frequency histograms of the parameters to be predicted. Finally, the cleaned data was exported to CSV to be loaded into Jupyter Notebook via Anaconda Navigator.

5 Implementation

This section provides a guide to the implementation of our research project, detailing every stage from loading the data into Jupyter Notebook through Anaconda Navigator using Python to train, validate, and test the models.

The first part consisted of the initial set up for our machine learning project, focused on a classification problem. All required libraries and modules for the project were imported in the first section of the code, as illustrated in Figure 6. It then the dataset was read from a CSV file and the data types of each feature were checked.

National College of Ireland

MSc in Data Analytics

Research Project

Groundwater Quality Predictive Analysis Using Machine Learning Techniques: Ireland

Leslie Rebeca Monroy Ochoa x23169761

In [1]:
```
#Importing all the required libraries for our project
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import xgboost as xgb
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix
from sklearn.preprocessing import LabelEncoder
```

Figure 6. Required Libraries Python

5.1 Groundwater Quality Classification

Our first experiment was to conduct a classification based on a set of interim guideline values for various groundwater quality parameters used in Ireland to classify whether a sample requires further testing or not. Figure 7 shows the function that examines every parameter; if any of the parameters are more than the interim guideline value, the function marked the row as “Further Action Required”. After, the classification was applied to our dataset, resulting in a new column that shows whether each sample meets the requirements or needs further testing.

In [6]:
```
#Implement classification function using only interim guideline values
def classify_water_quality(row):
    interim_guideline_values = {
        'Ammonia-Total (as N)': 0.15,
        'Calcium - filtered': 200,
        'Chloride': 30,
        'Total Hardness (as CaCO3)': 200,
        'Magnesium - filtered': 50,
        'Nitrate (as NO3)': 25,
        'Potassium - filtered': 5,
        'Sodium - filtered': 150,
        'Sulphate': 200
    }

    for param, limit in interim_guideline_values.items():
        if row[param] > limit:
            return 'Further action required'
    return 'All parameters within limits'

#Apply the classification
data['Groundwater_Quality'] = data.apply(classify_water_quality, axis=1)

#Display the first few rows with the new classification
data[['Groundwater_Quality']].head()
```

Out[6]:

	Groundwater_Quality
0	Further action required
1	Further action required
2	Further action required
3	Further action required
4	Further action required

Figure 7. Groundwater Classification

5.1 Decision Tree Classifier

Figure 8. shows the implementation of the Decision Tree Classifier. It starts with the target variables definition (Alkalinity, Dissolved Oxygen, Conductivity and Nitrate) that the models

attempted to predict. After the target variables were categorised into three quantiles and the features (X) and targets (Y) were defined. The data was then split into train and test sets using an 80:20 ratio. And the parameter grid was set up using max_depth, min_samples_split, min_samples_leaf and criterion into various possible configurations as shown below in Figure 8.

Decision Trees Classifier - With all the target variables

```
In [12]: #select our target variables
target_variables = [
    'Alkalinity-total (as CaCO3)',
    'Dissolved Oxygen (on-site)...7',
    'Conductivity @25°C',
    'Nitrate (as NO3)'
]

#binning the target variables into categories
y_binned = filtered_data[target_variables].apply(
    lambda x: pd.qcut(x, q=3, labels=['low', 'medium', 'high'])
)

#Define the features (X) and target (y)
X = filtered_data.drop(target_variables, axis=1)
y = y_binned

#Split the data into training and testing sets 80:20 ratio
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#Parameter grid for hyperparameter tuning
param_grid = {
    'estimator__max_depth': [3, 5, 7, 10, None],
    'estimator__min_samples_split': [2, 5, 10],
    'estimator__min_samples_leaf': [1, 2, 4],
    'estimator__criterion': ['gini', 'entropy']
}
```

Figure 8. DTC

Subsequently, the model was initialized and hyperparameter tuning was executed to obtain the best parameters to be used to reinitialize the classifier. Predictions were made in the training set. And finally, the model was evaluated using the selected performance metrics on the test set, computing each metric for every target value as presented in Figure 9.

```
#sTART the Decision Tree classifier
clf = MultiOutputClassifier(DecisionTreeClassifier(random_state=42))

#GridSearchCV with cross-validation
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid, cv=5, scoring='f1_weighted', n_jobs=-1)

#Fitting the model to the training data
grid_search.fit(X_train, y_train)

#Obtain the best parameters
best_params = grid_search.best_params_

#Train DTC With the best parameters
best_estimator = grid_search.best_estimator_
best_clf = MultiOutputClassifier(DecisionTreeClassifier(**best_estimator.estimator.get_params()))
best_clf.fit(X_train, y_train)

#Make predictions
y_pred_best = best_clf.predict(X_test)

#Combine the predictions into a DataFrame for display
predictions_df = pd.DataFrame(y_pred_best, columns=target_variables)
print(predictions_df.head())

#Model evaluation
evaluation_metrics = {}
for target in target_variables:
    accuracy = accuracy_score(y_test[target], y_pred_best[:, target_variables.index(target)])
    precision = precision_score(y_test[target], y_pred_best[:, target_variables.index(target)], average='weighted')
    recall = recall_score(y_test[target], y_pred_best[:, target_variables.index(target)], average='weighted')
    f1 = f1_score(y_test[target], y_pred_best[:, target_variables.index(target)], average='weighted')
    evaluation_metrics[target] = {'accuracy': accuracy, 'precision': precision, 'recall': recall, 'f1': f1}

for target, metrics in evaluation_metrics.items():
    print(f"Metrics for {target}:")
    print(f"  Accuracy: {metrics['accuracy']}")
    print(f"  Precision: {metrics['precision']}")
    print(f"  Recall: {metrics['recall']}")
    print(f"  F1 Score: {metrics['f1']}")
```

Figure 9. DTC Train and Evaluation

5.1 Random Forest Classifier

The implementation of the three pending models utilised the variables created during the DTC modelling process such as target_variables and independent_variables. The

implementation of the RFC is shown in the figure 10 below, starts with the creation of hyperparameter grid to tune parameters when the model is being trained, some of the parameters were: the number of trees in the forest, the maximum depth of each tree in the forest and the criterion to measure the quality of a split as shown in Figure 10 below. The model was trained using an 80:20 split. Next each target variable was used to evaluate the predictions.

```
#Start RFC
clf_rf = MultiOutputClassifier(RandomForestClassifier(random_state=42))

#Start GridSearchCV with cross-validation for Random Forest
grid_search_rf = GridSearchCV(estimator=clf_rf, param_grid=param_grid_rf, cv=3, scoring='f1_weighted', n_jobs=-1)

#Fitting the model
grid_search_rf.fit(X_train, y_train)

#Obtaing the best for RFC
best_params_rf = grid_search_rf.best_params_

#Train the RFC
best_estimator_rf = grid_search_rf.best_estimator_
best_clf_rf = MultiOutputClassifier(RandomForestClassifier(**best_estimator_rf.estimator.get_params()))
best_clf_rf.fit(X_train, y_train)

#Make predictions
y_pred_rf = best_clf_rf.predict(X_test)

#RFC model evaluation
evaluation_metrics_rf = {}
for target in target_variables:
    accuracy = accuracy_score(y_test[target], y_pred_rf[:, target_variables.index(target)])
    precision = precision_score(y_test[target], y_pred_rf[:, target_variables.index(target)], average='weighted')
    recall = recall_score(y_test[target], y_pred_rf[:, target_variables.index(target)], average='weighted')
    f1 = f1_score(y_test[target], y_pred_rf[:, target_variables.index(target)], average='weighted')
    evaluation_metrics_rf[target] = {'Accuracy': accuracy, 'Precision': precision, 'Recall': recall, 'F1 Score': f1}

#PRint the results
for target, metrics in evaluation_metrics_rf.items():
    print(f"Metrics for {target}:")
    print(f"  Accuracy: {metrics['Accuracy']}")
    print(f"  Precision: {metrics['Precision']}")
    print(f"  Recall: {metrics['Recall']}")
    print(f"  F1 Score: {metrics['F1 Score']}")
```

Figure 10. RFC Implementation

5.2 Support Vector Machine (SVM) Classifier

SVM model starts with the definition of a parameter grid for C and the kernel coefficient. The function used a 3-fold cross-validation to find the ideal parameters. An 80:20 ratio was then used to split the data into training and test sets respectively. The SVM was then trained using the optimal hyperparameters to continue with the model assessment on the test set. Finally, the metrics were printed to be evaluated. Illustrated in Figure 11 below.

```
#Parameter grid for hyperparameter tuning for SVM
param_grid_svm = {
    'estimator__C': [0.1, 1],
    'estimator__gamma': ['scale'],
}

#Start SVM classifier
clf_svm = MultiOutputClassifier(SVC(random_state=42))

#GridSearchCV with cross-validation for SVM
grid_search_svm = GridSearchCV(estimator=clf_svm, param_grid=param_grid_svm, cv=3, scoring='f1_weighted', n_jobs=-1)


# Split the data into training and testing sets (80:20) with the binned target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

#Fitting the SVM model to the training Set
grid_search_svm.fit(X_train, y_train)

#Training the SVM classifier
best_estimator_svm = grid_search_svm.best_estimator_
best_clf_svm = MultiOutputClassifier(SVC(**best_estimator_svm.estimator.get_params()))
best_clf_svm.fit(X_train, y_train)

#Make predictions
y_pred_svm = best_clf_svm.predict(X_test)

#SVM model evaluation
evaluation_metrics_svm = {}
for target in target_variables:
    accuracy = accuracy_score(y_test[target], y_pred_svm[:, target_variables.index(target)])
    precision = precision_score(y_test[target], y_pred_svm[:, target_variables.index(target)], average='weighted')
    recall = recall_score(y_test[target], y_pred_svm[:, target_variables.index(target)], average='weighted')
    f1 = f1_score(y_test[target], y_pred_svm[:, target_variables.index(target)], average='weighted')
    evaluation_metrics_svm[target] = {'Accuracy': accuracy, 'Precision': precision, 'Recall': recall, 'F1 Score': f1}

#Print the metrics for each target varieble
for target, metrics in evaluation_metrics_svm.items():
    print(f"Metrics for {target}:")
    print(f"  Accuracy: {metrics['Accuracy']}")
    print(f"  Precision: {metrics['Precision']}")
    print(f"  Recall: {metrics['Recall']}")
    print(f"  F1 Score: {metrics['F1 Score']}")
```

Figure 11. SVM Implementation

5.3 Extreme Gradient Boosting (XGBoost) Classifier

The last of the models that was implemented was XGBoost, for this specific model label encoding was required, followed by dataset splitting utilising an 80:20. After that, a grid of hyperparameters was implemented. The best parameters were used to train the model, and predictions were made on the test set. Finally, the model was evaluated using the selected performance metrics for each target variable, shown in Figure 12.

```
Metrics for Alkalinity-total (as CaCO3):
  Accuracy: 0.8046826863832409
  Precision: 0.8056693820054416
  Recall: 0.8046826863832409
  F1 Score: 0.8040010114945235
Metrics for Dissolved Oxygen (on-site)...7:
  Accuracy: 0.8835489833641405
  Precision: 0.8901924777933463
  Recall: 0.8835489833641405
  F1 Score: 0.8847539028217707
Metrics for Conductivity @25°C:
  Accuracy: 0.8422674060382008
  Precision: 0.8434927810974757
  Recall: 0.8422674060382008
  F1 Score: 0.8428106569425307
Metrics for Nitrate (as NO3):
  Accuracy: 0.9562538508934073
  Precision: 0.9607381516496583
  Recall: 0.9562538508934073
  F1 Score: 0.9567683822754817
```

Figure 12. XGBoost Metrics

Feature importances was also validated, using a bar plot, as presented in Figure 13.

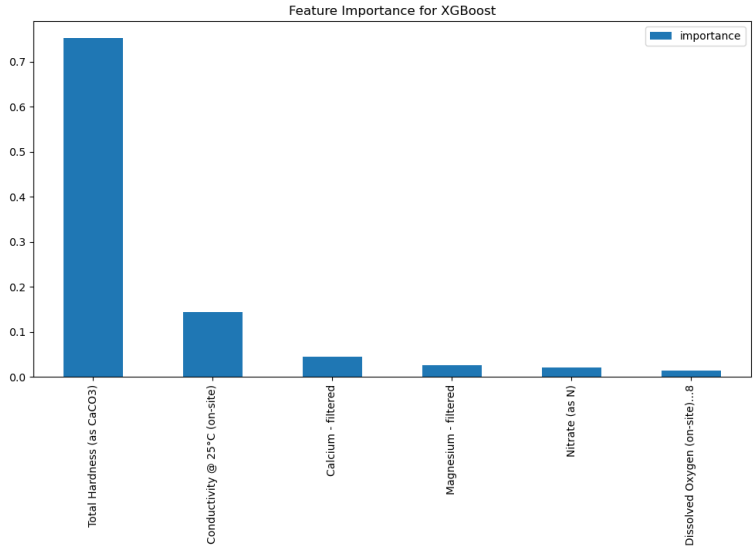

Figure 13. XGBoost Feature Importance

6 Conclusion

This configuration manual was created to be a useful tool to users and researchers to make sure that every step of project setup and implementation is comprehended and executed effectively. The complete code, visualisations, images and artifacts generated can be found for future references in the project documentation.