|  |
| --- |
| COMP 4254 CRN 85466 |
| Predictive Model for Drinking Water Quality |
| Assignment 2 |

|  |
| --- |
| Jacqueline Lee  A01295775 |

Table of Contents

[Introduction 1](#_Toc130932812)

[Exploratory Data Analysis and Data Preparation 2](#_Toc130932813)

[Data Overview 2](#_Toc130932814)

[Binning Variables According to Water Safety Guidelines 2](#_Toc130932815)

[Variable Correlation with Potability 3](#_Toc130932816)

[Feature Selection 4](#_Toc130932817)

[Model Development 6](#_Toc130932818)

[Scaling 6](#_Toc130932819)

[Base Model Refining for Stacking 6](#_Toc130932820)

[Chosen Models 8](#_Toc130932821)

[Model Evaluation 8](#_Toc130932822)

[Future Improvements 9](#_Toc130932823)

[Conclusion 9](#_Toc130932824)

[References 10](#_Toc130932825)

[Model Script 11](#_Toc130932826)

[Appendix - Exploration and Prep 15](#_Toc130932827)

# Introduction

Access to safe drinking water is a basic human right, but the safeness, or quality, of drinking water can be impacted by many different factors, both natural and manmade. As such, it is important to monitor as many of the factors as possible to determine the potability of water (or the ability to collect water from a body of water for consumption). The World Health Organization (WHO) has a set of guidelines for things to consider when monitoring water quality, and some of those features have been measured for this dataset. It is desired to build a model to predict potability from these features, not only to predict potability when given real samples, but also to provide insight as to where efforts should be focused to improve water treatment, both for waste and drinking water. Where certain features more negatively impact potability, initiatives can be made to improve treatment procedures to reduce the amounts of those contaminants.

From data exploration and preparation, a set of features are chosen based on significance and importance to the classifier models that are used in building the model using several different feature selection methods. From there, models are developed, starting from a basic logistic regression model to including stacking multiple classifier models and cross-fold validation. Performance metrics accuracy, precision, recall, and f1 scores are recorded for each fold, and evaluated using the average and standard deviation across the folds for performance and stability over multiple runs. The Receiver Operating Characteristic (ROC) curve and the area under the curve (ROC AUC) score are also recorded for the stacked model, to examine how well the models perform with respect to classifying Potability correctly.

The resulting model that is chosen at the end both performs relatively well and is relatively stable compared to the other models that are evaluated, and uses a mix of features that are significant and close-to-significant according to chi-square scoring in a stacked Logistic Regression model built off of 6 base classifier models (Logistic Regression, Decision Tree Classifier, Random Forest Classifier, AdaBoost Classifier, Gradient Boost Classifier, and XGB Classifier).

This dataset is more focused on the chemical aspect of water quality, which includes some of the factors that are prioritized in water treatment plants (especially in rural areas, where resources are limited, both with respect to material resources and human resources i.e. staffing and training). However, other factors in the WHO guidelines include those in the microbial, radiological, and acceptability aspects of water quality. Improvements to the model could focus on gathering measurements from those factors, as well as adjustments to the actual model development process, including using different methods for categorizing variables that span larger ranges and better refining the list of base models used to build stacked models.

# Exploratory Data Analysis and Data Preparation

## Data Overview

The potability of drinking water can be determined by many factors, from the different aspects of water quality: chemical, microbial, radiological, and acceptability (appearance, taste, smell, etc.). This dataset focuses mainly on the chemical aspects of water quality, with metrics such as those monitored by water treatment plants around the world. There are 9 key water quality metrics in this dataset, as seen in Table 1, all numerical, with potability being the target feature (the 10th metric) indicating whether the body of water surveyed is potable and safe for consumption based on the values of the other features.

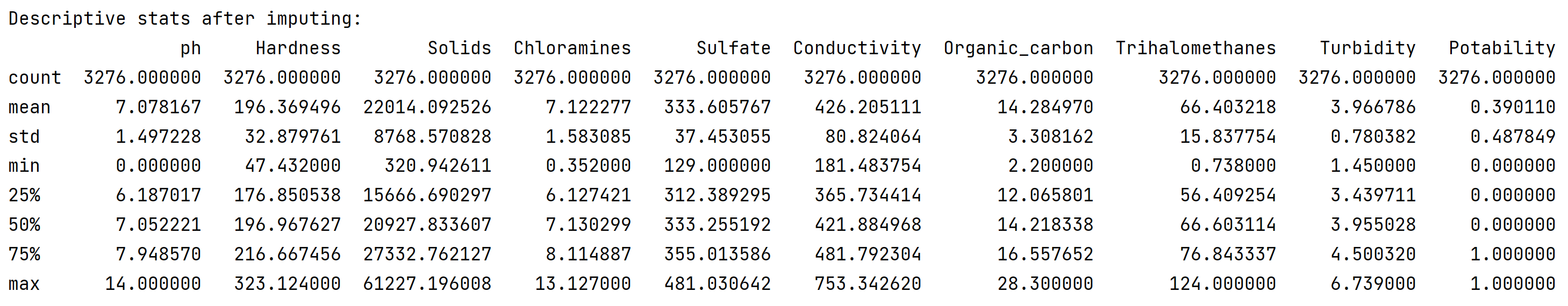
Table 1: Description of Water Quality Metrics in the Dataset [1, 2, 3]

|  |  |
| --- | --- |
| **Water Quality Metric** | **Description** |
| pH | Acidic/alkaline condition indicator. WHO guidelines state a permissible range of pH from 6.5 to 8.5. |
| Hardness | Capacity of water to precipitate soap, originally defined as being caused by calcium and magnesium content in water, in mg/L. |
| Solids (Total Dissolved Solids – TDS) | Dissolved inorganic and organic minerals and salts. WHO guidelines state a desirable limit of 500 mg/L and a maximum limit of 1000 mg/L for drinking purposes. |
| Chloramines | Product of adding ammonia to chlorine when treating drinking water. Safe limits for drinking water are typically set to 4 mg/L. |
| Sulfate | Naturally occurring substances in minerals, soil, and rocks; also widely used in the commercial chemical industry. |
| Conductivity (Electrical Conductivity – EC) | Measurement of the ionic process of a solution that enables it to transmit current. WHO guidelines state a permissible limit of 400 μS/cm. |
| Organic Carbon (Total Organic Carbon – TOC) | Total amount of carbon in organic compounds in water, in mg/L. |
| Trihalomethanes | By-product of treating natural organic matter with chlorine, in ppm. |
| Turbidity | Quantity of solid matter present in the water in a suspended state, measured by testing the light emitting properties of water, in NTU. |

There are 3276 total records in the dataset. The features Ph, Sulfate, and Trihalomethanes are missing values, so the K-Means nearest neighbour algorithm is used to impute the missing values. This algorithm takes the specified number of neighbours (in this case, 5), finds that many similar samples that have a value for the missing attribute, averages the values, and uses that result to fill in the missing value.

## Binning Variables According to Water Safety Guidelines

Looking at the descriptive statistics of the full dataset (including imputed columns), as seen in Figure 1, it is seen that some of the features span a much wider range than others – most notably, Solids, Sulfate, Conductivity, Hardness, and Trihalomethanes. Having variables with such varying ranges can affect the model and model performance, so binning some of these features was explored to see what effect it would have on feature significance.



*Figure 1: dataset descriptive statistics after imputing columns missing values [Appendix – Exploration and Prep].*

The features chosen for binning each had different safety limits that were explored when binning the categories. An explanation of each of the bins and their safety limits is given in Table 2. Safety limits are chosen for categorizing the variables as being a potential relevant factor to the dataset, especially when considering the fact that Potability is determined by acceptable and safe limits of contaminants and bacteria, among other factors in the real world.

Table 2: Binned Variables and Bin Descriptions

|  |  |
| --- | --- |
| **Binned Variable** | **Bins** |
| Solids | * Solids Desirable: values up to the desirable limit (500 mg/L) * Solids Safe: values up to the maximum limit (1000 mg/L) * Solids Unsafe: values exceeding the max limit |
| Chloramines | * Chloramines Safe: values up to the safe drinking level (4 mg/L) * Chloramines Unsafe: values exceeding the safe level |
| Conductivity | * Conductivity Safe: values up to the safe limit (400 μS/cm) * Conductivity Unsafe: values exceeding the limit |
| Trihalomethanes | * THM Safe: values up to the safe drinking level (80 ppm) * THM Unsafe: values exceeding the safe level |

## Variable Correlation with Potability

Correlation values are computed between each of the water quality metrics (binned and original columns included) and the target variable, Potability, to see if any of the variables are more strongly correlated with Potability (positively or negatively) and therefore potentially more significant to the model. However, none of the variables are very strongly correlated with Potability, with all correlation values falling between -0.05 and 0.05, as seen in Figure 2.

A picture containing table

Description automatically generated Graphical user interface, application, table

Description automatically generated

*Figure 2: correlation values between each feature and Potability. Figure on the left includes Potability, figure on the right shows just the predictor variables, to more clearly show which features are more positively or negatively correlated with Potability comparatively.*

The most positively correlated features include Solids, Chloramines, and Solids Safe, indicating that higher values in those variables contribute to more Potability. The most negatively correlated features include Organic Carbon and Sulfate, indicating that higher values in those variables contribute to less Potability.

It should be noted that the variables are not completely independent, and can be affected by other variables, as mentioned briefly in the variable descriptions in Table 1. Most notably, conductivity can be affected by the amount of dissolved solids (or ions) present in the sampled water, and the amount of trihalomethanes, as a by-product of chlorine treatment of water with organic carbon present in it, is impacted by both. Additionally, it should be noted that this means that THMs are typically inversely proportional with chloramines and TOC, as THM levels increase the more chlorine is used to treat higher organic carbon content (i.e. the more chlorine is used in reaction to remove organic content from water, the more THMs are produced). This is also consistent with the THM Unsafe variable being positively correlated with Potability and THM Safe being negatively correlated with Potability.

## Feature Selection

Several different methods are used to look at feature significance and importance to a classifier model, those methods being forward feature selection, recursive feature elimination, chi-square scoring, and the random forest classifier’s feature importance ranking, shown in Table 3. Forward feature selection and recursive feature elimination are automated feature selection methods, where the list of feature is iteratively added to (forward) or removed from (recursive) in order of significance to the model – in this case, Logistic Regression. Chi-square scoring calculates the chi-square scores of each of the features, with chi-square scores over 3.8 normally being considered significant to the model. The Random Forest Classifier algorithm is used to provide feature importance rankings, which help narrow down which features are most important to the model.

Table 3: Most Significant and Important Features According to Different Feature Selection Methods (chosen features are highlighted to show their respective position in the results of each method)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Forward Feature Selection** | | **Recursive Feature Elimination** | **Chi-Square Scoring** | | **Random Forest Feature Importance Ranking** | |
| *Feature* | *f-stat* | *Feature* | *Feature* | *Score* | *Feature* | *Importance* |
| Solids | 3.73 | Chloramines | Solids | 13023.98 | Sulfate | 0.13 |
| Organic\_carbon | 2.95 | Sulfate | Sulfate | 6.73 | Ph | 0.12 |
| Chloramines | 1.85 | Organic\_carbon | Hardness | 3.45 | Hardness | 0.11 |
| Sulfate | 1.60 | ConductivitySafe | Conductivity | 3.32 | Solids | 0.11 |
| SolidsSafe | 1.56 | THMSafe | Organic\_carbon | 2.26 | Chloramines | 0.11 |

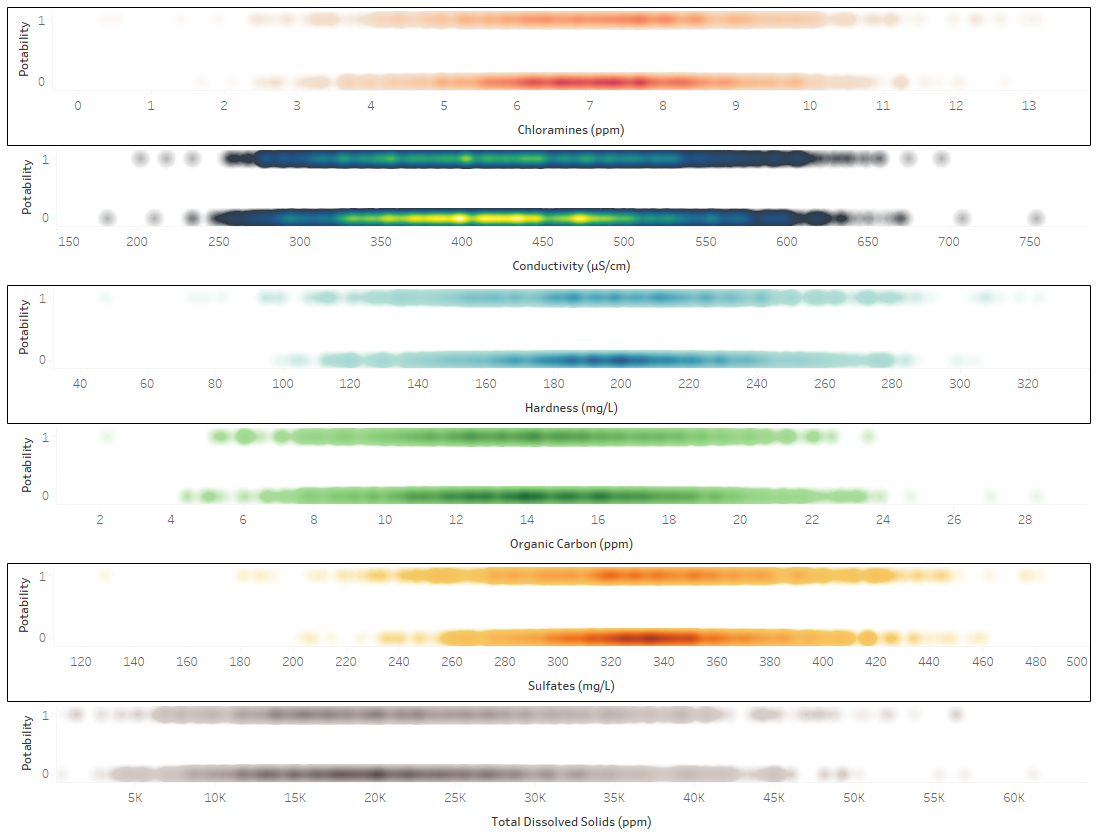
Scaling of the data is attempted to see if that would improve chi-square scoring, but only the MinMax Scaler is usable with the data (other methods throw errors due to calculations being computed on differing ranges between variables), and that proves to actually worsen the chi-square scores, as seen in Table 4. Therefore, scaling of data is left to model development, with only selected features being scaled for use in building the model.

Table 4: Chi-Square Scores of Scaled and Unscaled Features

|  |  |  |  |
| --- | --- | --- | --- |
| **Unscaled Features** | | **Scaled Features** | |
| *Feature* | *Chi-Square Score* | *Feature* | *Chi-Square Score* |
| Solids | 13023.98 | SolidsSafe | 1.56 |
| Sulfate | 6.73 | SolidsDesirable | 0.64 |
| Hardness | 3.45 | Solids | 0.22 |
| Conductivity | 3.32 | Organic\_carbon | 0.10 |

Likewise, since binned variables did not appear at the top of any of the lists of significant or important features, according to the feature selection methods’ results, binning is not carried out in the data preparation for the actual model.

Only a few variables are significant enough to include in the model, according to chi-square scores – Solids and Sulfate (Hardness and Conductivity are considered on the edge of significance for the purpose of model development). However, according to the automated feature selection methods, it appears that Organic\_carbon and Chloramines are also of some importance. To examine these features further, density plots relative to Potability are created for each feature, seen in Figure 3.



*Figure 3: density plots for each important feature (Chloramines, Conductivity, Hardness, Organic\_carbon, Sulfate, and Solids) with respect to Potability.*

The features appear to be similarly distributed across the density plots, corresponding with what is seen in the correlation heat maps in Figure 2 (where variables do not seem to be strongly correlated with Potability). This proves to make improving model performance difficult, since none of the features appear to significantly affect Potability according to the available data. An overview of different model feature sets is shown in Table 5, chosen with respect to the results of feature selection.

Table 5: Overview of Model Feature Sets

|  |  |  |
| --- | --- | --- |
| **Feature Set Label** | **Description** | **Features List** |
| **Basic** | All features considered important/significant enough to include in the model, excluding Conductivity (which had the lowest chi-square score)  Also used for scaled models | Solids, Sulfate, Hardness, Organic\_carbon, Chloramines |
| **Only Sig** | Features that were considered significant according to chi-square scoring (score > 3.8) | Solids, Sulfate |
| **Mixed** | Features that were considered significant according to chi-square scoring, including those that were on the edge of significance  Also used to trial stacked model refining | Solids, Sulfate, Hardness, Conductivity |
| **Auto** | Features that were considered significant according to the automated feature selection methods | Solids, Organic\_carbon, Sulfate, Chloramines |

# Model Development

Model development is carried out as follows: a basic logistic regression is first built, stacking of different base models is then implemented to create a better performing stacked model, before cross-fold validation is used to validate the stacked models across several folds. For each stacked model developed, the final model (built on the predictions of the base models) is chosen for performance metrics, which are compared between model algorithms. Stacked models are built off of a selection of the following base model algorithms: Logistic Regression, Decision Tree Classifier, Random Forest Classifier, XGB (Xtreme Gradient Boosting) Classifier, Adaptive Boosting Classifier, and Gradient Boosting Classifier. Cross-fold validation is carried out when building the models to be cross-evaluated, using the average and standard deviation of performance metrics across 3 folds.

## Scaling

The Basic feature set is used to see how scaling impacts model performance using the MinMax, Standard, and Robust scalers, with the best-performing model being chosen for model evaluation. Model performance metrics are shown in Table 6.

Table 6: Performance Metrics for Scaled and Unscaled Stacked Basic Model (best metric highlighted)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Features**: Solids, Sulfate, Hardness, Organic\_carbon, Chloramines  **Base Models for Stacking**: Logistic Regression, Decision Tree Classifier, Random Forest Classifier, XGB Classifier, Adaptive Boosting Classifier, and Gradient Boosting Classifier | | | | |
|  | **Unscaled** | **MinMax Scaled** | **Standard Scaled** | **Robust Scaled** |
| **Stacking Model** | AdaBoost Classifier | Random Forest | Logistic Regression | AdaBoost Classifier |
| **Accuracy – avg** | 0.67667 | 0.68333 | 0.67667 | 0.67000 |
| **Accuracy – std dev** | 0.00471 | 0.00943 | 0.01247 | 0.00816 |
| **Precision – avg** | 0.60333 | 0.66333 | 0.58333 | 0.60000 |
| **Precision – std dev** | 0.04784 | 0.02494 | 0.02494 | 0.04967 |
| **Recall – avg** | 0.20000 | 0.20000 | 0.19333 | 0.13333 |
| **Recall – std dev** | 0.05354 | 0.03742 | 0.03771 | 0.00471 |
| **F1 – avg** | 0.29667 | 0.30333 | 0.29333 | 0.22000 |
| **F1 – std dev** | 0.06342 | 0.03682 | 0.04497 | 0.01414 |
| **ROC AUC Score** | 0.545 | 0.557 | 0.595 | 0.591 |
| **ROC Curve** | Chart, line chart  Description automatically generated | Chart, line chart  Description automatically generated | Chart, line chart  Description automatically generated |  |

From examining the performance metrics, it appears that the MinMax Scaled version of the Basic Model, built using the Random Forest Classifier as the stacked model on top of the base models, is the best-performing (highest averages), most stable (lowest standard deviation) version of the models. Therefore, this model is chosen for evaluation against the other chosen models.

## Base Model Refining for Stacking

Another model that is tested is a refined version of the stacked model built using the Mixed feature set, which is the best performing out of the unscaled stacked models. Base model performance metrics are evaluated across the folds to build the stacked model, as seen in Table 7, and the worst-performing (lowest average), least stable (highest standard deviation) base models are removed from the list.

Table 7: Base Model Performance Metrics for the Mixed Stacked Model (worst-performing metrics highlighted in red, 2nd-worst-performing metrics highlighted in yellow)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Features**: Solids, Sulfate, Hardness, Conductivity | | | | | | |
|  | **Logistic Regression** | **Decision Tree Classifier** | **AdaBoost Classifier** | **Gradient Boost Classifier** | **XGB Classifier** | **Random Forest Classifier** |
| **Accuracy – avg** | 0.65667 | 0.58333 | 0.64667 | 0.67000 | 0.63000 | 0.64333 |
| **Accuracy – std dev** | 0.03300 | 0.02625 | 0.02055 | 0.01633 | 0.01414 | 0.02494 |
| **Precision – avg** | 0.58333 | 0.41000 | 0.51000 | 0.58333 | 0.46000 | 0.48667 |
| **Precision – std dev** | 0.42492 | 0.02160 | 0.06481 | 0.10077 | 0.03742 | 0.02867 |
| **Recall – avg** | 0.00667 | 0.43667 | 0.18667 | 0.23000 | 0.32333 | 0.27667 |
| **Recall – std dev** | 0.00471 | 0.03859 | 0.04110 | 0.05099 | 0.04110 | 0.05907 |
| **F1 – avg** | 0.01667 | 0.42333 | 0.27000 | 0.32333 | 0.37333 | 0.34667 |
| **F1 – std dev** | 0.01247 | 0.01247 | 0.03266 | 0.05249 | 0.03091 | 0.03771 |

From examining the averages and standard deviations of performance metrics across the different base models, the worst-performing base models are determined to be Logistic Regression and Decision Tree Classifier. As such, these models are removed from the list of base models, and the resulting model is evaluated against the original with respect to average performance metrics, as seen in Table 8.

Table 8: Refined and Original Mixed Stacked Model Performance Metrics

|  |  |  |
| --- | --- | --- |
| **Features**: Solids, Sulfate, Hardness, Conductivity | | |
|  | **Original** | **Refined** |
| **Base Models** | * Logistic Regression * Decision Tree Classifier * Random Forest Classifier * XGB Classifier * Adaptive Boosting Classifier * Gradient Boosting Classifier | * Random Forest Classifier * XGB Classifier * Adaptive Boosting Classifier * Gradient Boosting Classifier |
| **Stacking Model** | Logistic Regression | AdaBoost Classifier |
| **Accuracy – avg** | 0.68667 | 0.66333 |
| **Accuracy – std dev** | 0.03300 | 0.00471 |
| **Precision – avg** | 0.66000 | 0.57333 |
| **Precision – std dev** | 0.02160 | 0.05249 |
| **Recall – avg** | 0.18667 | 0.11133 |
| **Recall – std dev** | 0.01886 | 0.07074 |
| **F1 – avg** | 0.29000 | 0.24333 |
| **F1 – std dev** | 0.02160 | 0.02625 |
| **ROC AUC Score** | 0.567 | 0.539 |
| **ROC Curve** |  |  |

Examining the results show that the unrefined base model list provides better performance, so the Original Mixed stacked model is chosen for evaluation against the other chosen models.

## Chosen Models

Each of the feature sets are chosen to be evaluated in a stacked model against each other. The Basic feature set is chosen for cross-evaluation in the form of a MinMax Scaled model, while the chosen versions of the other stacked models are otherwise unmodified. The list of models to be evaluated, thus, is as follows: Only Sig, Mixed, Auto, and MinMax Scaled (Basic).

# Model Evaluation

Model development gives 4 models to be cross-evaluated with respect to performance metrics. The performance metrics of the stacked model are recorded across the folds during cross-fold validation for average and standard deviation calculations, the ROC (Receiver Operating Characteristic) curve is plotted for the final stacked model (as evaluated against the validation set), and the ROC AUC (Area Under the Curve) score is computed from the ROC curve. These metrics and scores are shown in Table 9 for cross-evaluation.

Table 9: Performance and ROC Metrics Across 3 Folds (best performing averages and standard deviations highlighted in green, 2nd best values highlighted in yellow)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Base Models for Stacking**: Logistic Regression, Decision Tree Classifier, Random Forest Classifier, XGB Classifier, Adaptive Boosting Classifier, and Gradient Boosting Classifier | | | | |
|  | **Only Sig** | **Mixed** | **Auto** | **MinMax Scaled** |
| **# of Features** | 2 | 4 | 4 | 5 |
| **Features** | * Solids * Sulfate | * Solids * Sulfate * Hardness * Conductivity | * Solids * Organic\_carbon * Sulfate * Chloramines | * Solids * Sulfate * Hardness * Organic\_carbon * Chloramines |
| **Stacking Model** | XGB Classifier | Logistic Regression | Logistic Regression | Random Forest |
| **Accuracy – avg** | 0.67333 | 0.68667 | 0.66333 | 0.68333 |
| **Accuracy – std dev** | 0.00943 | 0.03300 | 0.02867 | 0.00943 |
| **Precision – avg** | 0.65000 | 0.66000 | 0.36667 | 0.66333 |
| **Precision – std dev** | 0.02160 | 0.02160 | 0.25940 | 0.02494 |
| **Recall – avg** | 0.12333 | 0.18667 | 0.05667 | 0.20000 |
| **Recall – std dev** | 0.01247 | 0.01886 | 0.04190 | 0.03742 |
| **F1 – avg** | 0.20667 | 0.29000 | 0.10000 | 0.30333 |
| **F1 – std dev** | 0.01700 | 0.02160 | 0.07257 | 0.03682 |
| **ROC AUC Score** | 0.558 | 0.567 | 0.541 | 0.557 |
| **ROC Curve** | Chart, line chart  Description automatically generated | Chart, line chart  Description automatically generated |  | Chart, line chart  Description automatically generated |

Examining the results, it is observed that while the MinMax Scaled model performs better, it is not as stable as the Only Sig model. The Mixed model, on the other hand, both performs relatively well while also being relatively stable, and is thus chosen as the model to predict Potability.

# Future Improvements

It should be noted that the dataset used is suspiciously void of any time stamps or location tags for the stated 3276 different bodies of water surveyed. The distributions of the different metrics (with respect to the potability metric) are also very similar to each other. This was not caught until later in the model development stage, at which point the dataset was treated as more of a learning exercise. However, this dataset does provide a good look at the different metrics that are used in water treatment plants, especially those built in more rural parts of the world, where resources are difficult to deliver and the training and maintenance need to be kept simple to allow the people living there the ability to manage the plant without constantly calling in specialists.

A good start to future improvements would be, besides picking a dataset with real values tied to real locations and times, to find a dataset with more features included. Many of the different aspects to water quality are left out of this dataset, especially those involving the microbial and radiological aspects. Additionally, continuing to refine the stacked models by removing low-performing base models, categorizing the variables with larger ranges into smaller bins or by different values (such as variable quartiles provided by the dataset statistics), and increasing the number of folds in cross-fold validation might help improve the model.

# Conclusion

The potability of water is an important metric in determining whether water is safe for consumption and is impacted by many different features, including those grouped in microbial, radiological, acceptability, and chemical aspects of water quality. The dataset that is used to model potability focuses mostly on factors from the chemical aspects of water quality, namely those that are monitored by water treatment plants around the world. This model is built with the purpose of predicting potability for given measurements of different variables, but also with the purpose of indicating where initiatives could be made to improve water treatment efforts, to lessen the amounts of those contaminants that negatively impact potability.

The data is prepared and explored to determine which features are most significant to the model, and whether scaling and categorizing features improves model performance. The best-performing features are used to develop the model. Model development is done in steps, starting with a simple logistic regression. Stacking is then implemented, building a stacked model off of predictions made by a set of base models. Cross-fold validation is then used to assess the performance of the models in development across multiple runs. Several modifications to the models are tested, namely scaling of predictor variables and refining the list of base models used to build the stacked model. The chosen models are then evaluated, with the best-performing, most stable model proving to be the model using features selected using a mix of significant and close-to-significant features according to chi-square scoring (Solids, Sulfate, Hardness, and Conductivity), and built with a Logistic Regression model on the predictions of 6 classifier models (Logistic Regression, Decision Tree Classifier, Random Forest Classifier, AdaBoost Classifier, Gradient Boost Classifier, and XGB Classifier).

# References

[1] Dataset: <https://www.kaggle.com/datasets/adityakadiwal/water-potability>

[2] Geneva: World Health Organization (2022). Guidelines for drinking-water quality: fourth edition incorporating the first and second addenda. World Health Organization. Retrieved March 27, 2023, from <https://www.who.int/publications/i/item/9789240045064>.

[3] Department of Environment and Climate Change (2009, April 1). Reducing or eliminating Trihalomethanes (thms) in drinking water: Drinking water. Government of Nova Scotia - Department of Environment and Climate Change. Retrieved March 27, 2023, from <https://novascotia.ca/nse/water/thm.asp>.

# Model Script

|  |
| --- |
| *# Model Development with Cross-Fold Validation # Using features that are significant/somewhat significant # according to chi-square scores # ------------------------------------------------------------------------ # Loading libraries* import pandas as pd import numpy as np import matplotlib.pyplot as plt *# For imputing* from sklearn.impute import KNNImputer *# Models* from sklearn.linear\_model import LogisticRegression from sklearn.tree import DecisionTreeClassifier from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier from xgboost import XGBClassifier from sklearn.ensemble import RandomForestClassifier *# For modeling and evaluating* from sklearn.model\_selection import train\_test\_split, KFold from sklearn.metrics import precision\_score, recall\_score, f1\_score, accuracy\_score from sklearn.metrics import roc\_auc\_score from sklearn.metrics import roc\_curve  *# ------------------------------------------------------------------------ # Loading in the data* PATH = "C:\\Users\\jleer\\Documents\\BCIT\\COMP 4254\\assignments\\assign 2\\data\\" FILE = "water\_potability.csv" dataset = pd.read\_csv(PATH+FILE)  *# ------------------------------------------------------------------------ # Imputing columns using the KNN imputing method* imputer = KNNImputer(n\_neighbors=5) *# Using the 5 closest neighbours* dataset = pd.DataFrame(imputer.fit\_transform(dataset),  columns = dataset.columns)  *# Printing the dataset # Display options* pd.set\_option('display.max\_columns', None) pd.set\_option('display.width', 1000) print("Data Overview (After Imputing)\n1st 5 rows of data:") print(dataset.head(5)) *# Printing the 1st 5 records* print("\nDescriptive stats:") print(dataset.describe()) *# Printing descriptive stats for each col  # ------------------------------------------------------------------------ # Setting up the data for modeling* y = dataset['Potability'] *# Target variable* X = dataset[['Solids', 'Hardness',  'Sulfate', 'Conductivity']] *# Predictor variables  # Printing the model data set* print("\nModelling Dataset\ny:") print(y.head(5)) print("\nValue counts for target variable, y:") print(dataset['Potability'].value\_counts()) print("\nX:") print(X.head(5))  *# ------------------------------------------------------------------------ # Setting up the functions to: # - retrieve a list of models to fit # - evaluate models # - fit base models # - fit stacked models # - function that combines fitting the base and stacked models by # calling the functions that do those things separately # - function to evaluate the models with validation data  # Function to retrieve a list of models to fit:* def getUnfitModels():  *# Initializing an empty list* models = list()   *# Appending logistic regr/classifier models to the list* models.append(LogisticRegression())  models.append(DecisionTreeClassifier())  models.append(AdaBoostClassifier())  models.append(GradientBoostingClassifier())  models.append(XGBClassifier())  models.append(RandomForestClassifier(n\_estimators=10))   *# Returning the list to the calling function* return models  *# Function to evaluate models:* def evaluateModel(y\_test, predictions, model):  *# Calculating model performance metrics* precision = round(precision\_score(y\_test, predictions, zero\_division=0),2)  recall = round(recall\_score(y\_test, predictions), 2)  f1 = round(f1\_score(y\_test, predictions), 2)  accuracy = round(accuracy\_score(y\_test, predictions), 2)   *# Printing the metrics* print("Precision:" + str(precision) + " Recall:" + str(recall) +\  " F1:" + str(f1) + " Accuracy:" + str(accuracy) +\  " " + model.\_\_class\_\_.\_\_name\_\_)  *# Function to fit the base models:* def fitBaseModels(X\_train, y\_train, X\_test, y\_test, models):  *# Initializing an empty dataframe* dfPredictions = pd.DataFrame()   print("Base:") *# Indicator for fold evaluations   # Fitting base model and store its predictions in dataframe  # by looping through the list of models* for i in range(0, len(models)):  models[i].fit(X\_train, y\_train)  predictions = models[i].predict(X\_test)  evaluateModel(y\_test, predictions, models[i])  colName = str(i)   *# Adding base model predictions to column of data frame* dfPredictions[colName] = predictions   *# Returning base model predictions and the base models* return dfPredictions, models  *# Function to fit the stacked model:* def fitStackedModel(X, y):  *# Using logistic regression to fit the stacked model* model = LogisticRegression()  model.fit(X, y)   *# Evaluating stacked model* print("Stacked:")  evaluateModel(y, model.predict(X), model)   *# Returning the model* return model  *# Function that handles fitting the base and stacked models* def fitAllModels(X,y):  *# Getting the list of unfit models* unfitModels = getUnfitModels()  *# Initializing empty objects for the base and stacked models* models = None  stackedModel = None   *# Setting up cross-fold 3 folds* kfold = KFold(n\_splits=3, shuffle=True)  count = 0  *# Converting y to dataframe* y = y.to\_frame()  *# Looping for each fold* for train\_index, test\_index in kfold.split(X):  *# Splitting the data for the fold* X\_train = X.loc[X.index.intersection(train\_index), :]  X\_test = X.loc[X.index.intersection(test\_index), :]  y\_train = y.loc[y.index.intersection(train\_index), :]  y\_train = np.array(y\_train).reshape(-1, 1) *# Reshaping the data* y\_train = np.ravel(y\_train)  y\_test = y.loc[y.index.intersection(test\_index), :]  y\_test = np.array(y\_test).reshape(-1, 1) *# Reshaping the data* y\_test =np.ravel(y\_test)   *# Fitting base and stacked models* dfPredictions, models = fitBaseModels(X\_train, y\_train, X\_test, y\_test, unfitModels)  stackedModel = fitStackedModel(dfPredictions, y\_test)   *# Printing the current fold* count += 1  print("K-fold: " + str(count) + "\n")   *# Returning the base and stacked models* return models, stackedModel  *# Function to validate the base and stacked models with the validation set* def evaluateBaseAndStackModelsWithUnseenData(X, y, models, stackedModel):  *# Evaluating base models with validation data* print("\nEvaluating base models:")  *# Initializing an empty dataframe* dfValidationPredictions = pd.DataFrame()  *# Looping for the length of the models list* for i in range(0, len(models)):  predictions = models[i].predict(X) *# Using the validation set (X) to make predictions* colName = str(i)  dfValidationPredictions[colName] = predictions *# Putting the predictions into the dataframe* evaluateModel(y, predictions, models[i]) *# Evaluating the predictions with the validation set (y)   # Evaluating stacked model with validation data  # Using the predictions from the base models to make predictions using the stacked model* stackedPredictions = stackedModel.predict(dfValidationPredictions)  print("\nEvaluating stacked model:")  evaluateModel(y, stackedPredictions, stackedModel) *# Evaluating the predictions  # Using the predictions from the base models to get probability estimates using the stacked model* stackedProb = stackedModel.predict\_proba(dfValidationPredictions)  auc = roc\_auc\_score(y, stackedProb[:, 1], )  print('Logistic: ROC AUC=%.3f' % (auc))  *# Calculating roc curves* lr\_fpr, lr\_tpr, \_ = roc\_curve(y, stackedProb[:, 1])  plt.plot(lr\_fpr, lr\_tpr, marker='.', label='Logistic')  plt.plot([0, 1], [0, 1], '--', label='No Skill')  plt.xlabel('False Positive Rate')  plt.ylabel('True Positive Rate')  plt.legend()  plt.show()  *# ------------------------------------------------------------------------ # Building the model  # Splitting data into train, test, and validation sets # Splitting X and y into testing and validation sets on a 30% testing/ # 70% training split, where the training set will be split when going # through cross-fold to build the stacked model* X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.70)  *# Calling the function to fit the base and stacked models* print("\nFitting Models") models, stackedModel = fitAllModels(X\_train, y\_train)  *# Calling the function to evaluate the models* print("\nEvaluating Models With Unseen Data (Validation Set)") evaluateBaseAndStackModelsWithUnseenData(X\_test, y\_test, models, stackedModel) |

# Appendix - Exploration and Prep

|  |
| --- |
| *# Exploratory Data Analysis and Data Prep # ------------------------------------------------------------------------ # Loading libraries* import pandas as pd import numpy as np *# For imputing* from sklearn.impute import KNNImputer *# For automated feature selection* from sklearn.feature\_selection import f\_regression *# Forward feature sel.* from sklearn.feature\_selection import RFE *# Recursive feature elim.* from sklearn.linear\_model import LogisticRegression from sklearn.feature\_selection import chi2 *# Chi-square feature sel.* from sklearn.feature\_selection import SelectKBest from sklearn.ensemble import RandomForestClassifier *# Feature importance  # ------------------------------------------------------------------------ # Loading in the data* PATH = "C:\\Users\\jleer\\Documents\\BCIT\\COMP 4254\\assignments\\assign 2\\data\\" FILE = "water\_potability.csv" dataset = pd.read\_csv(PATH+FILE)  *# Printing the dataset # Display options* pd.set\_option('display.max\_columns', None) pd.set\_option('display.width', 1000) print("Data Overview\n1st 5 rows of data:") print(dataset.head(5)) *# Printing the 1st 5 records* print("\nDescriptive stats:") print(dataset.describe()) *# Printing descriptive stats for each col  # ------------------------------------------------------------------------ # Imputing columns using the KNN imputing method* imputer = KNNImputer(n\_neighbors=5) *# Using the 5 closest neighbours* dataset = pd.DataFrame(imputer.fit\_transform(dataset),  columns = dataset.columns)  *# Looking at the dataset after imputing* print("\nAfter Imputing\n1st 5 rows of imputed data:") print(dataset.head(5)) print("Descriptive stats after imputing:") print(dataset.describe())  *# ------------------------------------------------------------------------ # Exporting file to visualize imputed features as-is in Tableau # EXP\_FILE = "assign2\_ImpData.csv" # dataset.to\_csv(PATH+EXP\_FILE, header=True)  # ------------------------------------------------------------------------ # Binning feature variables according to water quality safety limits, as # specified by WHO  # Setting up the bins* dataset['SolidsBin'] = pd.cut(x = dataset['Solids'],  bins = [0, 500, 1000, 61230]) dataset['ChlorBin'] = pd.cut(x = dataset['Chloramines'],  bins = [0, 4, 14]) dataset['ECBin'] = pd.cut(x = dataset['Conductivity'],  bins = [0, 400, 755]) dataset['THMBin'] = pd.cut(x = dataset['Trihalomethanes'],  bins = [0, 80, 125])  *# Getting dummy variables for the bins # Isolating binned columns to retrieve dummies for* tempDF = dataset[['SolidsBin', 'ChlorBin', 'ECBin', 'THMBin']] *# Getting dummies for the isolated columns* dummyDF = pd.get\_dummies(tempDF, columns=['SolidsBin', 'ChlorBin', 'ECBin', 'THMBin']) *# Joining dummies back to the original dataset* dataset = pd.concat(([dataset, dummyDF]), axis=1)  *# Replacing dummy variable names with more easily understood names* dataset.rename(columns = {'SolidsBin\_(0, 500]': 'SolidsDesirable',  'SolidsBin\_(500, 1000]': 'SolidsSafe',  'SolidsBin\_(1000, 61230]': 'SolidsUnSafe',  'ChlorBin\_(0, 4]': 'ChlorSafe',  'ChlorBin\_(4, 14]': 'ChlorUnSafe',  'ECBin\_(0, 400]': 'ConductivitySafe',  'ECBin\_(400, 755]': 'ConductivityUnSafe',  'THMBin\_(0, 80]': 'THMSafe',  'THMBin\_(80, 125]': 'THMUnSafe'},  inplace = True)  *# Looking at the dataset after binning variables according # to WHO limits* print("\nAfter Binning\n1st 5 rows of binned data:") print(dataset.head(5)) print("Descriptive stats after binning:") print(dataset.describe())  *# ------------------------------------------------------------------------ # Looking at which columns are most strongly correlated with the target # variable, Potability, by exporting correlation values to visualize # in Tableau # Calculating correlation only for the numeric only columns (i.e. # excluding the binned variables and only taking their dummies)  # UNCOMMENT THE FOLLOWING TO EXPORT CORRELATION VALUES INTO A .CSV FILE # corr\_vals = dataset.corr(numeric\_only=True)[['Potability']].sort\_values(by='Potability',ascending=False) # EXP\_FILE = "assign2\_Exp\_CorrValues.csv" # corr\_vals.to\_csv(PATH+EXP\_FILE, header=True)  # ------------------------------------------------------------------------ # Identifying significant features using automated feature selection, # chi-square scores, and feature importance  # Separating the target variable from the predictors* y = dataset['Potability'] X = dataset.copy() *# Removing non-numerical variables (the bins) and the target # from the predictor set* X = X.drop(columns = ['Potability', 'SolidsBin',  'ChlorBin', 'ECBin', 'THMBin'])  *# Checking what the dataset looks like* print("\nAutomated Feature Selection") print("Separated data - target and predictors") print("Target:") print(y.head(5)) print("Predictors:") print(X.head(5))  *# Foward Feature Selection # Using f\_regression to retrieve the F-stat for each feature* ffs = f\_regression(X, y)  *# Putting each feature and its corresponding F-stat into a df # Initializing an empty dataframe* featuresDf = pd.DataFrame() *# Looping through the df for the length of the predictors set # to append each feature and its corresponding F-stat to the df* for i in range(0, len(X.columns)):  featureObject = {"feature":X.columns[i], "ffs":ffs[0][i]}  featuresDf = pd.concat([featuresDf, pd.DataFrame.from\_records([featureObject])]) *# Sorting the df by F-stat in descending order, as higher values # imply that the corresponding features are better for the model* featuresDf = featuresDf.sort\_values(by=['ffs'], ascending=False)  *# Looking at the results of Forward Feature Selection* print("\nForward Selected Features and FFS:") print(featuresDf)  *# Recursive Feature Elimination # Creating the model object* model = LogisticRegression(solver='liblinear') *# Using RFE to retrieve the specified number of features # that are most important to the model, according to RFE* rfe = RFE(model, n\_features\_to\_select=5) *# 5 out of 18 total features  # Putting the remaining features (after RFE) into an array # Initializing an empty list* features = [] *# Fitting the model with the data* rfe = rfe.fit(X, y) *# Checking what the results are* print('\nRecursive Feature Eliminated Features:') print(rfe.support\_) *# Appending the retrieved features to the list and checking # the result (with the print function)* columns = list(X.keys()) for i in range(0, len(columns)):  if(rfe.support\_[i]): *# i.e. if that index's feature is important according to RFE* print(columns[i])  features.append(X.keys()[i]) *# Appending the top features to the list of features  # Chi-Square Feature Selection # Using SelectKBest to get chi-square score for the specified # number of features that are most important to the model* test = SelectKBest(score\_func=chi2, k=10) *# Looking for 10 out of 18 total features # Note: chi-square score of >= 3.8 is generally considered # alright/significant enough to keep for the model* chiScores = test.fit(X, y) *# Summarize scores* np.set\_printoptions(precision=3)  *# Putting each feature and its corresponding chi-square score # into a df # Initializing an empty dataframe* dfFeatures = pd.DataFrame() *# Looping through the df for the length of the chi-square scores set to # append each feature and its corresponding chi-square score to the df* for i in range(0, len(chiScores.scores\_)):  headers = list(X.keys())  featureObject = {"feature":headers[i], "chi-square score":chiScores.scores\_[i]}  dfFeatures = pd.concat([dfFeatures, pd.DataFrame.from\_records([featureObject])])  *# Looking at the chi-square scores of predictor features* print("\nPredictor Variables' Chi-Square Scores:") dfFeatures = dfFeatures.sort\_values(by=['chi-square score'], ascending=False) print(dfFeatures)  *# Random Forest Feature Importance # Creating a Gaussian Classifier* rf=RandomForestClassifier(n\_estimators=100) *# Fitting the model with the data* rf = rf.fit(X, y) *# Getting feature importances* importances = list(rf.feature\_importances\_) *# Creating a list of tuples with variable and importance* features\_list = list(X.keys()) feature\_importances = [(feature, round(importance, 2)) for feature, importance in zip(features\_list, importances)] *# Sorting the feature importances by most important first* feature\_importances = sorted(feature\_importances, key = lambda x: x[1], reverse = True)  *# Looking at the importance of features for Random Forest Classifier* print("\nFeature importance for the random forest:") [print('Variable: {:20} Importance: {}'.format(\*pair)) for pair in feature\_importances] |