Predicting Water Potability Using Machine Learning Techniques

Milestone 4: Perform your own experiments

Team members:

1. Mahesh Jayaraman (mj456)
2. Jyothsna Kaamala (jk734)
3. Himasundhar Raju Meesala (hm496)

Submission Date: April 25, 2023

Course Name: DS675 Machine Learning

Instructor's Name: Michael Houle

Video Presentation Link:
https://drive.google.com/drive/folders/11rellI2eXuHwbLgw6eqmFY9c-xjytbRpT?usp=share_link

Colab Link:
https://colab.research.google.com/drive/1E-GZaTfRkhttU7qySqI4KBEKU-OTa3wH

Objective:

The primary objective of this project is to develop a machine learning model that accurately predicts water potability based on a set of physicochemical parameters. This will help in identifying safe drinking water sources and guiding necessary treatment processes.

Dataset description:

The Kaggle water potability dataset contains 3,276 water quality samples. Each sample is described by nine physicochemical properties, including pH, hardness, solids content, etc. The dataset also includes a binary target variable indicating if the water is safe to drink (1) or (0). The dataset was sourced from Aditya Kadiwal.

Rows: 3,276
Columns: 10

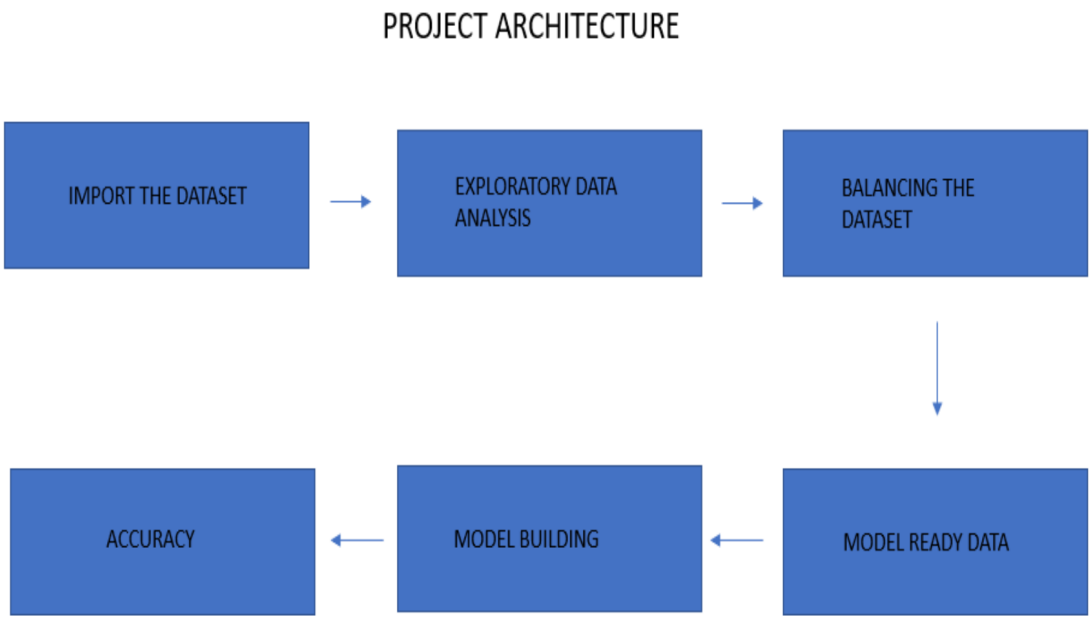
Usability: 10.0

Dataset link: https://www.kaggle.com/datasets/adityakadiwal/water-potability

Aims:

The project aims to achieve the following sub-objectives:

1. Perform exploratory data analysis (EDA) to understand the dataset's features, their relationships, and distributions.
2. Preprocess the data by cleaning it, handling missing values, and normalizing or standardizing the features.
3. Address imbalances in the target variable by balancing the number of classes and using techniques such as Synthetic Minority Oversampling Technique (SOMTE).
4. Split the dataset into training and testing sets for model evaluation and validation.
5. Experiment with various machine learning algorithms, including K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Logistic Regression, AdaBoost, and Bagging Classifier, to find the best performing model for the given dataset.



Division of duties:

The project responsibilities are divided among team members as follows:

1. Mahesh Jayaraman:
    a. Address imbalances in the target variable using SMOTE
    b. Perform EDA
    c. Visualize and present the results of different models
    d. Implement Logistic Regression model and perform Cross-validation
2. Jyothsna Kaamala:
    a. Data cleaning visualizations
    b. Split the dataset into training and testing sets
    c. Implement machine learning models, including KNN and Bagging classifier.
3. Himasundhar Raju Meesala:
    a. Handle missing values in the dataset
    b. Implement machine learning models including SVM, and Adaboost.

Dataset exploration:

Information on the data

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

We can clearly see that certain columns contain null values.

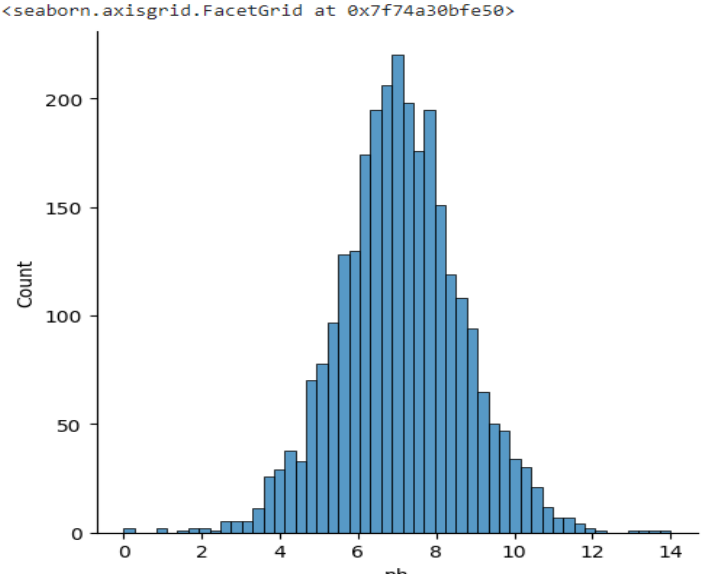
```
[ ]  df.isna().sum()
```

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

We plot ph, sulfate, and Trihalomethanes to check what method to use for imputing.

```
[10] sns.displot(df.ph)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f74a30bfe50>
```

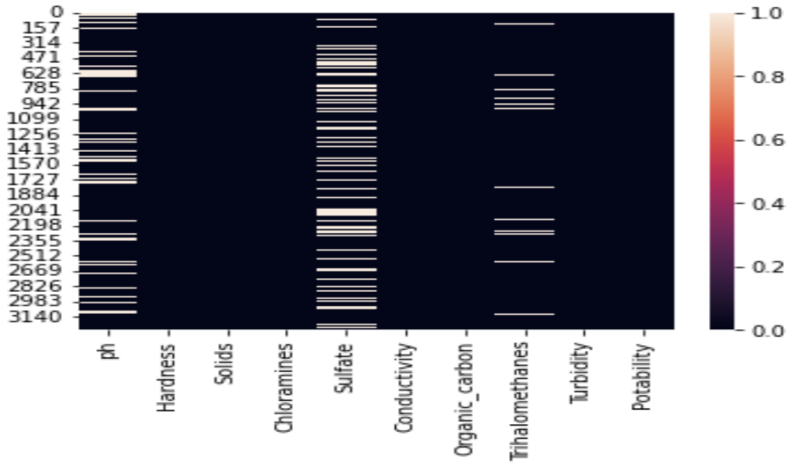


Above, we can see that the ph plot is normally distributed and similarly others columns are normally distributed too. We can use ‘mean’ to impute the values.

Data cleaning visualizations

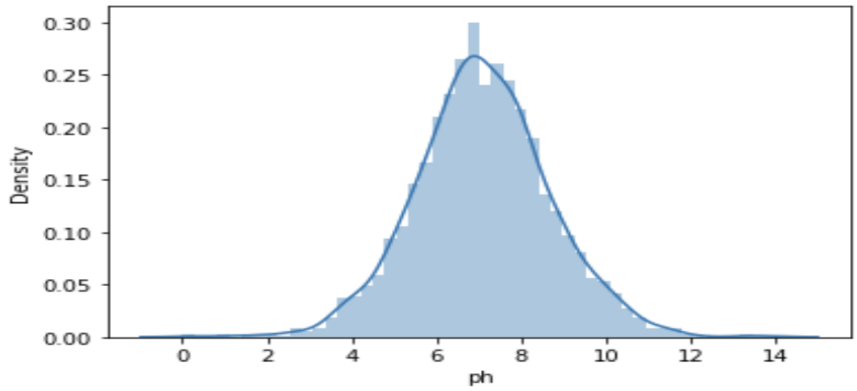
Identified null values using a heatmap for visualization, revealing missing values in the columns pH, Sulfate, and Trihalomethanes.

```
In [7]: sns.heatmap(data.isnull())
Out[7]: <AxesSubplot:>
```



Examined each column with null values individually and determined the appropriate measure of central tendency for imputation. Analyzed the ‘pH’ column with a distribution plot, observing an almost perfectly normal distribution.

```
In [12]: sns.distplot(data.ph)
Out[12]: <AxesSubplot:xlabel='ph', ylabel='Density'>
```



Imputed missing values in the ‘pH’ column with the mean due to the normal distribution. Followed similar steps for the ‘sulfate’ and ‘Trihalomethanes’ columns, analyzing their respective distributions and imputing missing values accordingly.

Balancing the class using SMOTE:

We tried looking into various techniques and decided to try balancing the data using SMOTE. Synthetic Minority Oversampling Technique (SMOTE) does nothing but oversample the minority class. The performance of a model typically depends on the minority class and we wanted to see how this would improve the model performance.

```
X_sm, y_sm = sm.fit_resample(X, y)

print(f'''Shape of X before SMOTE: {X.shape}
Shape of X after SMOTE: {X_sm.shape}''')

print('\nBalance of positive and negative classes (%):')
y_sm.value_counts(normalize=True) * 100
```

```
Shape of X before SMOTE: (3276, 9)
Shape of X after SMOTE: (3996, 9)

Balance of positive and negative classes (%):
0    50.0
1    50.0
Name: Potability, dtype: float64
```

After using SMOTE, we can see that the classes became balanced.

Split the dataset into training and testing sets

Split the feature matrix (X) and target vector (Y) into training and testing sets, enabling model evaluation and validation on unseen data.

```
In [48]: from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X_sm, y_sm, test_size=0.25, random_state=16
)
```

Modeling:

KNN

1. Fit the KNN model on the training set.
2. Make predictions on the unseen test dataset.
3. Determine the optimal k-value for the dataset using the following steps:
    a. Split the data into training and test sets.
    b. Loop over various k-values and evaluate their performance.
    c. Compute training and test data accuracy
    d. Generate a plot to visualize the relationship between k-values and accuracy.
    e. Examine the train and test split visualization.
4. Visualize predictions on the test set.
5. Assess probability estimates and model confidence:
    a. Create a mesh grid for running the model.
    b. Use the classifier to make predictions on the grid.
    c. Plot the results, comparing model confidence with true labels.
6. Apply the model to the dataset:
    a. Load and split the data.
    b. Create a mesh grid for running the model.
    c. Use the classifier to make predictions on the grid.

KNN

In [50]:
```
# Fitting model on the training and test dataset
knn = KNeighborsClassifier(n_neighbors=4, p=2, metric='minkowski')
knn.fit(X_train, y_train)

print('The accuracy of the knn classifier is {:.2f} out of 1 on training data'.format(knn.score(X_train, y_train)))
print('The accuracy of the knn classifier is {:.2f} out of 1 on test data'.format(knn.score(X_test, y_test)))
```

```
The accuracy of the knn classifier is 0.82 out of 1 on training data
The accuracy of the knn classifier is 0.67 out of 1 on test data
```

We have used the KNN and we got around 67% accuracy on the test data.

Logistic regression

We got a test accuracy of 85%. We tried crossfold validation (score. 81.25%) which did not improve the accuracy.

```
[51] from sklearn.linear_model import LogisticRegression
     logreg = LogisticRegression(random_state=42)
     logreg.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(random_state=42)
```

```
[52] y_pred = logreg.predict(X_test)
```

```
[53] accuracy = accuracy_score(y_test, y_pred)
     accuracy_percentage = 100 * accuracy
```

```
[54] accuracy_percentage
```

```
85.0
```

```
from sklearn.model_selection import KFold, cross_val_score
clf=LogisticRegression(random_state=42)

folds=KFold(n_splits=5)

scores=cross_val_score(clf, X_train,y_train,cv=folds)

print("cross val scores:",scores)
print("Average CV score:",scores.mean())
```

```
cross val scores: [0.875  0.8125 0.8125 0.6875 0.875 ]
Average CV score: 0.8125
```

AdaBoost

Then we fitted the model using ensemble methods (AdaBoost).
Here we have performed hyperparameter tuning to get the best parameters for a good model fit.

AdaBoost

```
In [83]:  1 ada = AdaBoostClassifier()
```

```
In [84]:  1 #Finding Best parameters for Adaboost Classifier
```

```
In [85]:  1 params_ada = {'n_estimators': [50,100,250,400,500,600], 'learning_rate': [0.2,0.5,0.8,1]}
          2 grid_ada =  GridSearchCV(ada, param_grid=params_ada, cv=5)
```

```
In [86]:  1 grid_ada.fit(X_train, y_train)
```

```
Out[86]: GridSearchCV(cv=5, estimator=AdaBoostClassifier(),
                      param_grid={'learning_rate': [0.2, 0.5, 0.8, 1],
                                  'n_estimators': [50, 100, 250, 400, 500, 600]})
```

```
In [87]:  1 print("Best parameters for AdaBoost:", grid_ada.best_params_)
```

```
Best parameters for AdaBoost: {'learning_rate': 0.8, 'n_estimators': 400}
```

SVM

We have used the SVM classifier and we got around 85% accuracy on the test data.

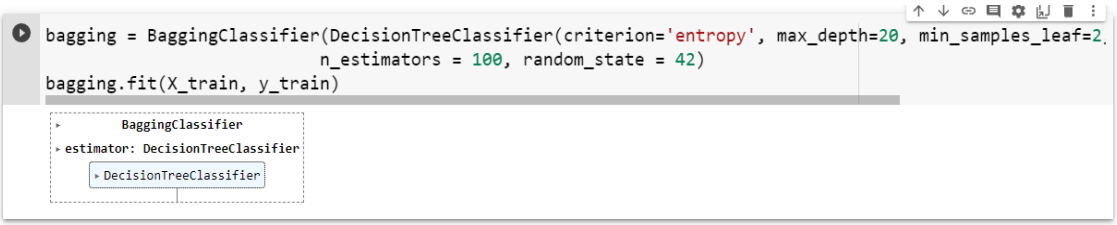
SVM

```
In [68]:
1 from sklearn.svm import SVC
2
3 svm = SVC(kernel='rbf', random_state=0, gamma=.10, C=1.0, probability=True)
4 svm.fit(X_train, y_train)
5
6 print('The accuracy of the svm classifier on training data is {:.2f} out of 1'.format(svm.score(X_train, y_train)))
7
8 print('The accuracy of the svm classifier on test data is {:.2f} out of 1'.format(svm.score(X_test, y_test)))
```

```
The accuracy of the svm classifier on training data is 0.82 out of 1
The accuracy of the svm classifier on test data is 0.85 out of 1
```

Bagging Classifiers

1. Create and fit a Bagging Classifier with DecisionTreeClassifier as the base estimator on the training data.
2. Make predictions using the Bagging Classifier on the test data.
3. Calculate and report the accuracy score by comparing predicted values to true values.




```
[ ] y_pred = bagging.predict(X_test)
    accuracy = accuracy_score(y_test,y_pred)
    accuracy
```

```
0.95
```

Results and Discussion

The following are the accuracy's that are observed with classification models built.

AdaBoost Classifier - 95.0 %

Bagging Classifier - 95.0 %

Logistic Regression - 85.0 %

SVM - 85.0 %

KNN - 67.0 %

We tried experimenting with various ML algorithms, including k-NN, SVM, Logistic Regression, AdaBoost and Bagging Classifier to see how well it can classify after balancing the dataset. Compared to other studies k-NN performance did not improve that much. The SVM performed significantly better than the previous results. Logistic Regression gave a similar accuracy to that of SVM and also significantly better than other works. AdaBoost and Bagging Classifier accuracy is 95% which is better than all the models we tried before. Comparing this to prior work, We can say that there have been significant results to our experiments. The most popular model for this dataset used by others is Random Forest, it gave an accuracy of 88%.

In **Milestone 2**, we did a brief literature review on 2 notebooks. The first notebook used spot checking and found the best model. They performed spot checking and decided the best performing model. They were fine tuned and the accuracy achieved was 66%. The second notebook took the imbalance in data into account. After basic modeling, XGB and bagging gave high accuracy. After fine tuning, bagging classification improved. They also tried H2o AutoML which performed the best and gave an accuracy of 80%. Even with these results if we compare our work, After SMOTE, our Bagging and AdaBoost model worked significantly better than the prior models.

References

1. https://www.kaggle.com/datasets/adityakadiwal/water-potability?datasetId=1292407&sortBy=voteCount
2. https://arxiv.org/pdf/1106.1813