Date of publication 07/29/2023

Digital Object Identifier 10.1109/ACCESS.2023.0322000

Exploring K-Nearest Neighbors (KNN) Algorithm for Water Quality Prediction

Amit Raj Pant[1] Arahanta Pokhrel[1]

Insitute of Engineering, Thapathali Campus, Final Year

Corresponding authors:
Amit Raj Pant (e-mail:[2]amitrajpant7@gmail.com),
Arahanta Pokhrel (e-mail:[1]pokhrelarahanta5@gmail.com).

ABSTRACT This report presents a comprehensive analysis of water quality data using the K-Nearest Neighbors (KNN) classifier. The dataset contains information on various water quality parameters, such as pH, hardness, solids, chloramines, sulfate, conductivity, organic carbon, trihalomethanes, turbidity, and potability. The primary objective of this analysis is to predict water potability based on these attributes using the KNN algorithm. The report discusses the methodology, data preprocessing, model training and evaluation, feature selection, and the comparison of results with different KNN models. We were able to get an accuracy of around 66%, with comparable accuracy to existing work. Additionally, insights obtained from the analysis and the significance of the experiment are discussed.

INDEX TERMS KNN classifier, water quality dataset, potability prediction

I. INTRODUCTION

Water quality is a critical factor that directly impacts public health, environmental sustainability, and economic development. Access to clean and safe drinking water is a fundamental human right, and ensuring its availability is a global challenge. The evaluation and prediction of water potability are essential to safeguard the well-being of communities and protect natural water resources.The objective of this lab is to analyze a comprehensive water quality dataset using the K-Nearest Neighbors (KNN) classifier. The dataset encompasses vital water quality attributes, including pH, hardness, solids, chloramines, sulfate, conductivity, organic carbon, trihalomethanes, turbidity, and potability. By employing the KNN algorithm, we aim to build predictive models that can classify water samples as potable or non-potable based on these attributes.The importance of water quality assessment cannot be overstated, especially considering the challenges posed by pollution, urbanization, and climate change. Unsafe water can lead to severe health consequences, including waterborne diseases and contaminant exposure. Consequently,we use set of methodology and deploy accurate and reliable predictive models to classify water sources as potable or non-potable.

The KNN classifier, a non-parametric and lazy learning algorithm, offers an attractive choice for this classification task. By considering the majority class of the nearest neighbors in the feature space, KNN adapts well to the underlying data distribution, making it suitable for diverse water quality datasets with varying complexities. This report explores the application of the KNN algorithm to classify water quality data and aims to identify the significant attributes that influence water potability.This report goes beyond merely applying the K-Nearest Neighbors (KNN) algorithm to predict water potability and delves into its theoretical foundations, offering a deeper understanding of its functioning and capabilities. It presents major mathematical formulae, such as the Euclidean distance metric, and showcases a system block diagram, providing a clear visualization of the data processing and model training steps. The methodology section details data preprocessing steps, major Python libraries used, and a step-by-step working principle. The results section conveys the analysis through plots, graphs, and tables, em-

phasizing insights on attributes significantly impacting water potability. we successfully applied the KNN classifier on the water quality dataset and gain knowledge about its potential as a powerful tool for water potability prediction.

II. METHODOLOGY

A. THEORY

The K-Nearest Neighbor (KNN) algorithm**[1]**, a popular machine learning technique, is widely used for classification and regression tasks. Its foundation lies in the notion that similar data points tend to have similar labels or values. During the training phase, KNN stores the entire training dataset as a reference for future predictions. When making predictions, it calculates the distance between the input data point and all the training examples using a chosen distance metric like Euclidean distance.

The next step involves identifying the K nearest neighbors to the input data point based on their calculated distances. In classification tasks, KNN assigns the most common class label among the K neighbors as the predicted label for the input data point. For regression, it computes the average or weighted average of the target values of the K neighbors to predict the value for the input data point. This way, KNN classifies data points based on their similarity to neighboring points and makes educated guesses about unclassified points.

KNN is classified as non-parametric and an example of lazy learning. Being non-parametric, the algorithm makes no assumptions about the underlying data distribution and constructs its model solely from the given data. It avoids assuming any normal structure, making it more flexible for various datasets. Additionally, KNN is considered lazy learning because it does not create an explicit model during the training phase. Instead, it stores the training data and uses it directly during testing, making little effort in the training process. This characteristic implies that KNN does not generalize from the training data, relying solely on the stored information during the prediction phase.

The KNN algorithm is straightforward and easy to understand**[2]**, making it a popular choice in various domains. However, its performance can be affected by the choice of K and the distance metric, so careful parameter tuning is necessary for optimal results.

In K-Nearest Neighbors (KNN) algorithm, the letter "K" represents the number of nearest neighbors considered for classification. When classifying a new data point, KNN looks at its K closest data points in the training set to determine its label. For example, if K=1, the test examples are given the same label as the closest example in the training set. If K=3, the algorithm checks the labels of the three closest data points and assigns the most common label (occurring at least twice) to the new data point.

Choosing the appropriate value for K is crucial when building a KNN classifier. You can either have a specific value in mind or use techniques like cross-validation to test various values of K and determine which one works best for your data. For a dataset with n=1000 cases, it is likely that the optimal value of K lies somewhere between 1 and 19. However, the best approach is to experiment with different K values to ensure the classifier's optimal performance for the specific dataset.

The KNN algorithm operates on the principle that similar data points tend to have similar labels. It calculates the distance between the new data point and all the existing data points in the dataset using a chosen distance metric, such as the Euclidean distance formula.

1) Euclidean distance

The Euclidean distance formula is a measure of the straight-line distance between two points (x, y) and (a, b) on a graph. It is given by the equation:

$$\text{dist}(d) = \sqrt{(x-a)^2 + (y-b)^2} \tag{1}$$

Where:

- x, y are the coordinates of the existing data point in the dataset.
- a, b are the coordinates of the new data points.
- $\text{dist}(d)$ represents the Euclidean distance between the two points.

For a higher-dimensional space, the formula extends as follows:

$$\text{dist}(d) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2}$$

where x_i and y_i are the values of the i-th feature in the two data points. By calculating the Euclidean distance between the new data point and all the existing data points, we can identify the k-nearest neighbors to the new data point based on their distances.

2) Distance Metrics:

In addition to the Euclidean distance, there are other distance metrics you can use in KNN. Some alternatives include:

- Manhattan Distance: The Manhattan distance calculates the sum of the absolute differences between corresponding feature values of two data points. For a multi-dimensional space with n features, the Manhattan distance between two points (x, y) is given by:

$$\text{dist}(d) = \sum_{i=1}^{n} |x_i - y_i| \tag{3}$$

where x_i and y_i are the values of the i-th feature in the two data points.

- Minkowski Distance: The Minkowski distance is a generalization of both the Euclidean and Manhattan distances and is controlled by a parameter p. For a multi-dimensional space with n features, the Minkowski distance between two points (x, y) is given by:

$$\text{dist}(d) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}} \tag{4}$$

where x_i and y_i are the values of the i-th feature in the two data points, and p is the parameter controlling the distance calculation. When $p = 2$, it becomes the Euclidean distance; when $p = 1$, it becomes the Manhattan distance.

- Cosine Similarity: The cosine similarity measures the cosine of the angle between two data points in a multi-dimensional space. It is commonly used for text data or documents. For a multi-dimensional space with n features, the cosine similarity between two points (x, y) is given by:

$$\text{dist}(d) = 1 - \frac{\sum_{i=1}^{n} x_i \cdot y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \cdot \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{5}$$

where x_i and y_i are the values of the i-th feature in the two data points. Cosine similarity ranges from -1 to 1, where 1 indicates that the vectors point in the same direction, 0 indicates orthogonality, and -1 indicates they point in opposite directions.

3) Standardization:

Standardizing the data involves scaling each feature to have zero mean and unit variance. The formula for standardizing a feature x is given by:

$$x' = \frac{x - \text{mean}(x)}{\text{std}(x)} \tag{6}$$

where x' is the standardized value of x, $\text{mean}(x)$ is the mean of feature x, and $\text{std}(x)$ is the standard deviation of feature x. Standardization helps in bringing all features to a common scale, preventing any single feature from dominating the distance calculations in KNN.

4) Advantages of KNN Algorithm

- The algorithm is simple and easy to implement.
- There's no need to build a model, tune several parameters, or make additional assumptions.
- The algorithm is versatile. It can be used for classification, regression, and search (as we will see in the next section).

5) Disadvantages of KNN Algorithm

- The algorithm gets significantly slower as the number of examples and/or predictors/independent variables increase.

B. DATASET

We plotted the heatmap of corelation between the features of our datase. In figure

The `water_potability.csv` file contains water quality metrics for 3276 different water bodies.

1) **pH value:**
   pH is an important parameter in evaluating the acid–base balance of water. It is also the indicator of acidic or alkaline condition of water status. WHO has recommended a maximum permissible limit of pH from 6.5 to 8.5. The current investigation ranges were 6.52–6.83 which are in the range of WHO standards.

2) **Hardness:**
   Hardness is mainly caused by calcium and magnesium salts. These salts are dissolved from geologic deposits through which water travels. The length of time water is in contact with hardness-producing material helps determine how much hardness there is in raw water. Hardness was originally defined as the capacity of water to precipitate soap caused by Calcium and Magnesium.

3) **Solids (Total Dissolved Solids - TDS):**
   Water has the ability to dissolve a wide range of inorganic and some organic minerals or salts such as potassium, calcium, sodium, bicarbonates, chlorides, magnesium, sulfates, etc. These minerals produce an unwanted taste and diluted color in the appearance of water. This is an important parameter for the use

of water. The water with high TDS value indicates that water is highly mineralized. The desirable limit for TDS is 500 mg/l, and the maximum limit is 1000 mg/l, which is prescribed for drinking purposes.

4) **Chloramines:**
Chlorine and chloramine are the major disinfectants used in public water systems. Chloramines are most commonly formed when ammonia is added to chlorine to treat drinking water. Chlorine levels up to 4 milligrams per liter (mg/L or 4 parts per million (ppm)) are considered safe in drinking water.

5) **Sulfate:**
Sulfates are naturally occurring substances found in minerals, soil, and rocks. They are present in ambient air, groundwater, plants, and food. The principal commercial use of sulfate is in the chemical industry. Sulfate concentration in seawater is about 2,700 milligrams per liter (mg/L). It ranges from 3 to 30 mg/L in most freshwater supplies, although much higher concentrations (1000 mg/L) are found in some geographic locations.

6) **Conductivity:**
Pure water is not a good conductor of electric current but rather a good insulator. An increase in ion concentration enhances the electrical conductivity of water. Generally, the amount of dissolved solids in water determines the electrical conductivity. Electrical conductivity (EC) actually measures the ionic process of a solution that enables it to transmit current. According to WHO standards, the EC value should not exceed 400 S/cm.

7) **Organic Carbon:**
Total Organic Carbon (TOC) in source waters comes from decaying natural organic matter (NOM) as well as synthetic sources. TOC is a measure of the total amount of carbon in organic compounds in pure water. According to the US EPA, < 2 mg/L as TOC in treated/-drinking water, and < 4 mg/L in source water which is used for treatment.

8) **Trihalomethanes:**
THMs are chemicals which may be found in water treated with chlorine. The concentration of THMs in drinking water varies according to the level of organic material in the water, the amount of chlorine required to treat the water, and the temperature of the water that is being treated. THM levels up to 80 ppm are considered safe in drinking water.

9) **Turbidity:**
The turbidity of water depends on the quantity of solid matter present in the suspended state. It is a measure of light-emitting properties of water, and the test is used to indicate the quality of waste discharge with respect to colloidal matter. The mean turbidity value obtained for Wondo Genet Campus (0.98 NTU) is lower than the WHO recommended value of 5.00 NTU.

10) **Potability:**
Indicates if water is safe for human consumption where 1 means Potable and 0 means Not potable.

C. WORKING PRINCIPLE

To initiate the analysis, an initial exploration of the dataset was undertaken. This exploration encompassed a thorough assessment of attribute correlations, achieved through correlation coefficient calculations. Histogram plots and kernel density estimation were also employed to gain insights into the distribution and density of data points. Concurrently, the dataset underwent a meticulous process to handle missing values, with the imputation strategy involving the replacement of missing entries with the mean value of the respective attribute.

Following the preliminary exploration and preprocessing, a pivotal step was the partitioning of the dataset into training and testing subsets, adopting an 80:20 ratio to ensure both sufficient training data for the model's learning and an appropriate volume of unseen instances for evaluation.

Subsequently, the KNN model was instantiated, marking a significant juncture in the analysis. The model was meticulously trained using the training dataset, which imparted the essence of the data distribution and patterns to the algorithm, enabling it to make informed predictions.

The predictive performance of the KNN model was rigorously assessed through a multifaceted evaluation approach, incorporating diverse metrics to capture various dimensions of model effectiveness. Metrics such as precision, recall, F1-score, and accuracy were meticulously calculated, each offering distinct insights into the model's behavior and effi-

cacy. To enhance the clarity and interpretability of the evaluation outcomes, a heatmap plot was crafted, visualizing the results in a comprehensive and insightful manner.

1) **Data Loading and Preprocessing:** The dataset is loaded from a CSV file into a pandas DataFrame named 'df'. Missing values are checked and replaced with column means using the 'replace_NAN' function. Descriptive statistics are computed to understand the data.

2) **Data Exploration and Visualization:** Initial data exploration is performed by displaying the first few rows of 'df'. The count of 'Potability' classes is visualized using a countplot, showing the distribution of "Not Drinkable" and "Drinkable" instances. Correlation matrix is computed, and a heatmap is plotted to visualize feature correlations. Histograms with KDE are plotted to understand the distributions of each feature.

3) **Model Building and Evaluation:** The `train_eval` function is defined to standardize the data, split it into training and testing sets, and train a K-Nearest Neighbors (KNN) classifier. The classifier's performance is evaluated using classification_report and confusion_matrix on the test data.

4) **Finding Optimal 'k' Value:** A list of 'k' values ranging from 1 to 51 (with a step of 2) is created. For each 'k', a KNN model is trained and evaluated using cross-validation with 10 folds. The accuracy scores are collected for each 'k' value and plotted against 'k' to find the optimal value.

5) **Feature Removal and Retraining:** The 'Sulfate' feature is dropped from the dataset, and the model is retrained and evaluated on the modified data.

6) **Tackling Class Imbalance:** To address class imbalance, 'Not Drinkable' instances are downsampled to match the number of 'Drinkable' instances. The balanced dataset is shuffled, and the KNN model is retrained and evaluated on it.

7) **Final Results:** The performance metrics and visualizations are generated for the balanced dataset after handling class imbalance. The methodology aims to build and evaluate an effective KNN classifier for predicting water potability.

D. INSTRUMENTATION

In this lab, the following major libraries and functions were used for implementing the Navie Bayes algorithm, performing data manipulation, and evaluating the results:

- `pandas`: Data manipulation and analysis, data frames creation, dataset loading, and preprocessing.
- `numpy`: Numerical computations, array operations, mathematical functions for data manipulation and preprocessing.
- `seaborn`: Data visualization, informative statistical graphics, high-level interface for visually appealing plots.
- `plotly.express`: Interactive and dynamic data visualization with customizable charts and plots.
- `sklearn.neighbors.KNeighborsClassifier`: K-Nearest Neighbors (KNN) algorithm for classification tasks. It finds the 'k' nearest neighbors to a data point and assigns the class label based on majority voting among the neighbors. KNN is suitable for both numerical and categorical features.
- `train_test_split`: Used for dataset splitting into training and test sets for model evaluation.
- `classification_report`: Generates a comprehensive report of classification results with metrics like precision, recall, F1-score, and support for each class.
- `confusion_matrix`: Computation of the confusion matrix for evaluating model performance and misclassifications.
- `matplotlib.pyplot`: Data visualization, creation of plots, graphs, and figures, including decision tree visualization.
- `sklearn.preprocessing.StandardScaler`: This module provides access to the StandardScaler class in scikit-learn, which allows us to standardize the features of a dataset
- `cross_val_score`: This module provides access to the `cross_val_score` function in scikit-learn, which facilitates cross-validation. It helps to evaluate the performance of a model by splitting the data into multiple folds, training the model on different subsets of the data, and calculating the evaluation metric (such as accuracy) on each fold.

By utilizing these libraries and functions, we were able to effectively implement the KNN algorithm, preprocess the data, split the dataset, train the models, evaluate their performance, and visualize the results in this lab.

E. SYSTEM METHODOLOGY

The system block diagram is represented in figure 1 in the appendix section.

III. RESULTS

A. DATASET ANALYSIS

The dataset exhibits a class imbalance, with undrinkable water instances outnumbering drinkable water instances. This imbalance could potentially lead to bias in predictions, favoring the overrepresented class. Addressing this bias may require specialized modeling techniques or evaluation metrics. In scenarios like these, where class imbalance is present, strategies such as resampling techniques (e.g., oversampling or undersampling) or utilizing appropriate evaluation metrics like precision-recall curves can be e mployed to mitigate potential bias and enhance the model's overall effectiveness. Furthermore, an initial analysis of the dataset revealed the presence of missing values in the Sulphur column. To mitigate this issue, the missing values were imputed with the mean value of the column.

An examination of attribute relationships indicates a minimal correlation, implying a predominantly independent nature of features. This independence poses challenges in constructing a K-Nearest Neighbors (KNN) model but also suggests limited potential for significant attribute elimination affecting overall model performance.

B. MODEL PERFORMANCE

The initial training of our model yielded an accuracy of only 61%. The performance of the model is summarized in Table 1. The model demonstrated commendable precision (0.66) and recall (0.80), leading to a well-balanced F1-score (0.72) when classifying unportable instances. However, its performance was subpar in identifying portable water instances, exhibiting lower precision (0.56) and recall (0.39), translating to a moderate F1-score (0.46). It is evident that enhancing the model's ability to detect portable instances could substantially boost its overall performance. An important contributing factor to these results could be the class imbalance within our dataset, compounded by the presence of missing values that were imputed with mean values, potentially compromising the dataset's integrity.

TABLE 1: Classification Report

Class	Precision	Recall	F1-Score	Support
0	0.66	0.80	0.72	391
1	0.56	0.39	0.46	265
Accuracy			0.63	
Macro Avg			0.59	
Weighted Avg			0.63	

To enhance model accuracy, a pivotal step involved dropping incomplete data points followed by training a new K-Nearest Neighbors (KNN) classifier. This led to measurable improvements in classification, as depicted in Table 2.

TABLE 2: Classification Report after Dropping Incomplete Data Points

Class	Precision	Recall	F1-Score	Support
0	0.67	0.83	0.74	233
1	0.65	0.43	0.52	170
Accuracy			0.66	403
Macro Avg	0.66	0.63	0.63	403
Weighted Avg	0.66	0.66	0.65	403

Following the removal of incomplete instances, precision for the "not portable water" class improved from 0.66 to 0.67. The recall rate also witnessed a notable enhancement, rising from 0.80 to 0.83, consequently yielding a higher F1-score of 0.74. This indicates that the model has become more adept at correctly identifying instances of "not portable water." Furthermore, precision for the "portable water" class experienced a significant upsurge, ascending from 0.56 to 0.65, thus showcasing enhanced accuracy in classifying samples as "portable." This improved performance is also reflected in the heatmap plot of the confusion matrix, as depicted in Figure 5.

IV. DISCUSSION AND FURTHER INSIGHTS

Incorporating cross-validation into our analysis revealed that the optimal value for k was determined to be 25. While the model did not exhibit consistent accuracy improvements with an increase in the number of neighbors, $k = 25$ yielded the highest accuracy, as illustrated in Figure 6.

Despite numerous attempts to enhance the model's accuracy, we were only able to achieve a modest improvement, reaching an accuracy of 67%. Our efforts included techniques such as balancing the dataset and experimenting with different strategies to handle missing values (mean, median, and manual imputation). However, the limited success could stem from inherent inconsistencies within the dataset and the insufficiency of attributes to robustly determine water drinkability.

In conclusion, while our model's accuracy showed some progress, the challenges posed by an imbalanced dataset and missing values underscore the complexity of accurately classifying portable and unportable water instances. Further exploration of feature engineering and potentially incorporating external data sources may be necessary to achieve more substantial improvements in model performance.

V. CONCLUSION

In conclusion, our analysis delved into the intricacies of the dataset, revealing class imbalance and attribute independence that posed challenges to model development. We recognized the significance of addressing bias in predictions due to class imbalance, advocating for specialized techniques like resampling and tailored evaluation metrics. Additionally, we highlighted the impact of missing values on model integrity and employed mean imputation to mitigate their effects.

The initial model performance underscored the need for improvement, with notable differences in precision, recall, and F1-score between the two classes. Through careful refinement, including dropping incomplete instances, our model demonstrated commendable progress. Precision, recall, and F1-score for both classes improved significantly, illustrating the potency of data preprocessing. The integration of cross-validation also provided insights into optimal parameter selection, exemplified by the determination of k=25 as the best choice for the K-Nearest Neighbors model.

Despite our concerted efforts, achieving a substantial accuracy boost remained elusive, likely due to inherent dataset inconsistencies and the intricate nature of determining water drinkability. Our journey illuminated the multifaceted nature of the problem and emphasized the importance of meticulous feature engineering and potentially incorporating external data sources for more substantial enhancements in model performance.

VI. REFERENCES

References

[1] Guo, Gongde; Wang, Hui; Bell, David; Bi, Yaxin. *KNN Model-Based Approach in Classification*. August 8, 2004.

[2] Zhang, Z. *Introduction to machine learning: k-nearest neighbors. Ann Transl Med.* 2016 Jun;4(11):218. doi: 10.21037/atm.2016.03.37. PMID: 27386492; PMCID: PMC4916348.

APPENDIX. FIGURES AND PLOTS

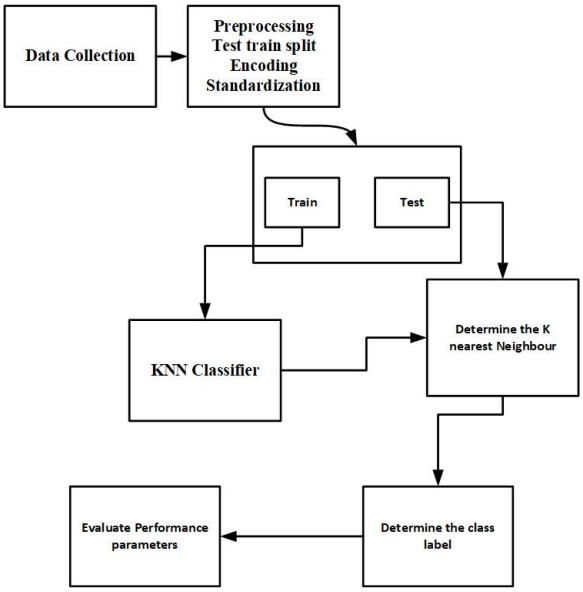

FIGURE 1: System Diagram

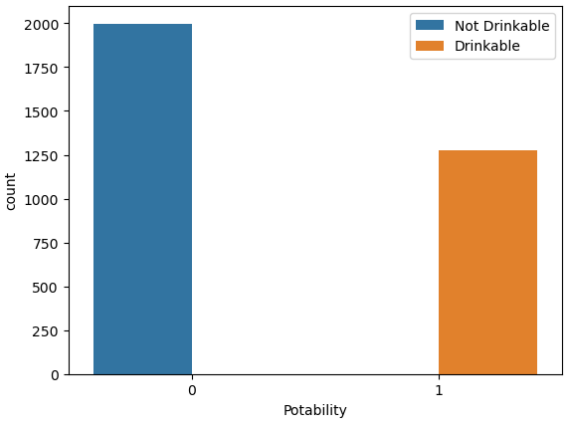

FIGURE 2: Class distribution

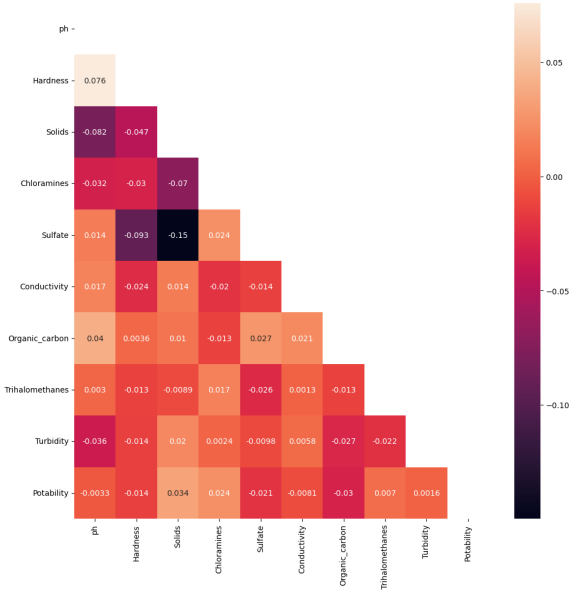

FIGURE 3: Correlation between attributes

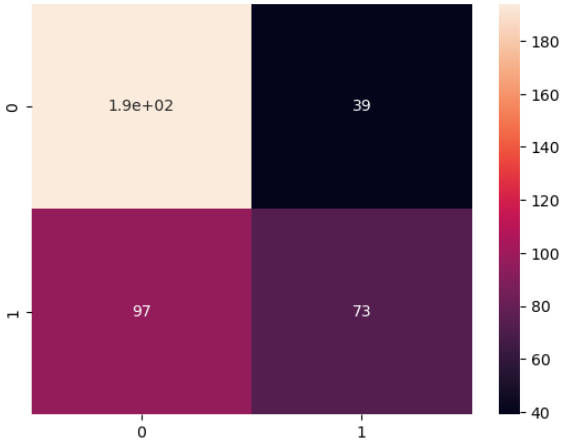

FIGURE 5: Heatmap plot of confusion matrix after using dropped dataset

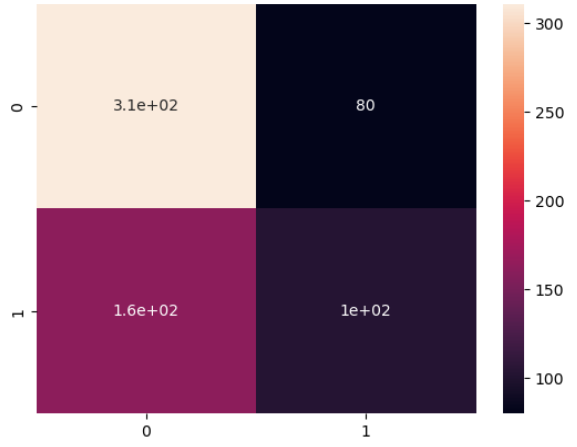

FIGURE 4: Heatmap plot of confusion matrix

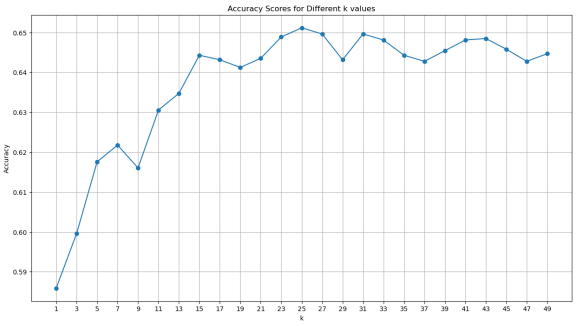

FIGURE 6: Heatmap plot of confusion matrix after using dropped dataset

APPENDIX. CODE

```

import numpy as np
import matplotlib.pyplot as plt
from sklearn.neighbors import
    KNeighborsClassifier
import pandas as pd
import seaborn as sns
from sklearn.preprocessing import
    StandardScaler
from sklearn.metrics import
    classification_report, confusion_matrix
from sklearn.model_selection import
    cross_val_score
from sklearn.model_selection import
    train_test_split

df = pd.read_csv('/kaggle/input/water-
    potability/water_potability.csv')


df.head()

missing_values = df.isnull().sum()
print(missing_values)

df.describe()

#Replaces empety values
def replace_NAN(d):
    missing_values = d.isnull().sum()
    means = d.mean()
    for col in d.columns:
        if missing_values[col] > 0:
            d[col].fillna(means[col],
                inplace=True)
    return d

means = df.mean()
print(means)

df = replace_NAN(df)

missing_values = df.isnull().sum()
print(missing_values)

df.describe()

#visiualizing the data and class
    distribution

sns.countplot(data=df, x= 'Potability', hue=
     'Potability')
plt.legend(labels= ['Not Drinkable', '
    Drinkable'])

cor = df.corr()
m = np.triu(cor)
plt.figure(figsize = (12,12))
sns.heatmap(cor, annot= True, mask =m)

for col in df.columns:
    sns.histplot(data=df, x=col, kde = 'True
        ',  hue= 'Potability')
#     plt.legend(labels= ['Not Drinkable', '
    Drinkable'])
    plt.show()



#Most of the attributes follow normal
    distribution

def train_eval(data):

    # Standardizing the dataset
    data = data.sample(frac=1)
    data_x= data.iloc[:, :-1]
    data_y = data.iloc[:,-1]
    scaler = StandardScaler()
    data_x = pd.DataFrame(scaler.
        fit_transform(data_x), columns =
        data_x.columns)

    #test train split
    x_train, x_test, y_train, y_test =
        train_test_split(data_x, data_y,
        test_size = 0.2, random_state=41)

    #model creation
    model = KNeighborsClassifier()
    model.fit(x_train, y_train)

    #evaluation
    predictions = model.predict(x_test)
    report = classification_report(y_test,
        predictions)
    print(report)
    matrix = confusion_matrix(y_test,
        predictions)
    sns.heatmap(matrix, annot= True)

train_eval(df)

#Poor accuracy, maybe due to imbalanced data

#Testing k-values
data=df
data = data.sample(frac=1)
data_x= data.iloc[:, :-1]
data_y = data.iloc[:,-1]
scaler = StandardScaler()
data_x = pd.DataFrame(scaler.fit_transform(
    data_x), columns = data_x.columns)

#test train split
x_train, x_test, y_train, y_test =
    train_test_split(data_x, data_y,
    test_size = 0.2, random_state=41)




k_values = list(range(1,51, 2))
accu = []
for k in k_values:
    m = KNeighborsClassifier(n_neighbors =k)
    scores = cross_val_score(m, x_train,
        y_train, cv=10,  scoring='accuracy'
        )
    accu.append(scores.mean())

# Plotting accuracy scores versus k values
plt.figure(figsize =(15,8))
plt.plot(k_values, accu, marker='o')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.title('Accuracy Scores for Different k
    values')
plt.grid(True)
plt.show()

#Try dropping sulphata colmn
df = df.drop(columns = 'Sulfate', axis =1 )
df.head(5)

train_eval(df)

#Lets tackel class imbalance problem, by
    downsampaling not drinkable instances(
    np)

p = data[data['Potability']==1]
not_p = data[data['Potability']==0]
sampled_data = not_p.sample(n=1300)

#Shuffle the sampled data
```

```
data_new = pd.concat([p, sampled_data], axis
    =0)
#shuffle
s_data = data_new.sample(frac=1)
s_data.head()

sns.countplot(data=s_data, x= 'Potability',
    hue= 'Potability')
# plt.legend(labels= ['Not Drinkable', '
    Drinkable'])



#visiualizing the data and class
    distribution

#lets train model
#replacing NAN
# s_data = replace_NAN(s_data)
train_eval(s_data)


# Lets try dropping all the rows with
    missing values
data = pd.read_csv('/kaggle/input/water-
    potability/water_potability.csv')
data.dropna(inplace =True)

sns.countplot(data=data, x = 'Potability',
    hue = 'Potability' )
print(data.shape)

train_eval(data)

data = data.sample(frac=1)
data_x= data.iloc[:, :-1]
data_y = data.iloc[:,-1]
scaler = StandardScaler()
data_x = pd.DataFrame(scaler.fit_transform(
    data_x), columns = data_x.columns)

#test train split
x_train, x_test, y_train, y_test =
    train_test_split(data_x, data_y,
    test_size = 0.2, random_state=41)




k_values = list(range(1,51, 2))
accu = []
for k in k_values:
    m = KNeighborsClassifier(n_neighbors =k)
    scores = cross_val_score(m, x_train,
        y_train, cv=10,  scoring='accuracy'
        )
    accu.append(scores.mean())

# Plotting accuracy scores versus k values
plt.figure(figsize =(15,8))
plt.plot(k_values, accu, marker='o')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.title('Accuracy Scores for Different k
    values')
plt.grid(True)
plt.show()

dataset = prep.skewcorrect(dataset,
    except_columns=['Potability'])

#balancing the class
data[data['Potability']==1].shape

#Lets tackel class imbalance problem, by
    downsampaling not drinkable instances(
    np)

p = data[data['Potability']==1]
not_p = data[data['Potability']==0]
sampled_data = not_p.sample(n=811)

#Shuffle the sampled data
data_new = pd.concat([p, sampled_data], axis
    =0)
#shuffle
s_data = data_new.sample(frac=1)
s_data.head()

sns.countplot(data=s_data, x= 'Potability',
    hue= 'Potability')
# plt.legend(labels= ['Not Drinkable', '
    Drinkable'])

train_eval(s_data)

data = s_data
data_x= data.iloc[:, :-1]
data_y = data.iloc[:,-1]
scaler = StandardScaler()
data_x = pd.DataFrame(scaler.fit_transform(
    data_x), columns = data_x.columns)

#test train split
x_train, x_test, y_train, y_test =
    train_test_split(data_x, data_y,
    test_size = 0.2, random_state=41)




k_values = list(range(1,51, 2))
accu = []
for k in k_values:
    m = KNeighborsClassifier(n_neighbors =k)
    scores = cross_val_score(m, x_train,
        y_train, cv=10,  scoring='accuracy'
        )
    accu.append(scores.mean())

# Plotting accuracy scores versus k values
plt.figure(figsize =(15,8))
plt.plot(k_values, accu, marker='o')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.xticks(k_values)
plt.title('Accuracy Scores for Different k
    values')
plt.grid(True)
plt.show()
```

APPENDIX. AUTHORS

ARAHANTA POKHAREL was born in 1999 in Biratnagar, Nepal. He is a dedicated individual with a strong passion for learning and research. He is currently pursuing a Bachelor's degree in Computer Technology at the Institute of Engineering, Thapathali Campus, in the final year of his studies. He has developed a keen interest in machine learning and data science throughout his academic journey. Additionally, he has a curious mind, actively engages in quizzes and current affairs to stay updated with the latest information, and is committed to acquiring new skills.

AMIT RAJ PANT Amit Raj Pant is a dedicated student pursuing his studies in the Department of Electronics and Computers at Thapathali Engineering Campus, Tribhuvan University, located in Kathmandu, Nepal. With a strong passion for technology, his interests include computer vision and machine learning on resource-constrained edge devices, which involves performing computational tasks on local devices rather than relying solely on remote servers.