COS40007 Artificial Intelligence for Engineering

Portfolio Assessment 1: "Hello Machine Learning for Engineering"

Student Name: Nguyen Quy Hung
Student Number: 104850199
Studio Class: Studio 1&2
Submission Date: September 14, 2025

1. Dataset Selection

Selected Dataset: Water Potability Dataset (water_potability.csv)

I selected a dataset comprising 3,276 water samples with 9 physicochemical parameters and 1 target variable indicating water potability. The original features include: pH, Hardness, Solids, Chloramines, Sulfate, Conductivity, Organic_carbon, Trihalomethanes, and Turbidity.

2. Rationale for Dataset Selection

I chose the Water Potability Dataset because it directly relates to environmental engineering applications, which aligns with my interests in data-driven solutions for real-world problems. As a third-year engineering student, I find water quality assessment particularly relevant since it represents a critical domain where machine learning can provide substantial value in automated monitoring systems. The binary classification problem of determining water safety based on quantitative chemical measurements reflects scenarios I might encounter in my future career, where data-driven approaches can complement traditional analytical methods in environmental monitoring and regulatory compliance.

---

3. Exploratory Data Analysis Summary

3.1 Dataset Overview

I began my analysis by examining the dataset structure and quality**:**

```python
# Initial data overview
print("INITIAL DATA OVERVIEW:")
print(f"Dataset shape: {df.shape}")
print(f"Columns: {list(df.columns)}")
print("\nFirst 5 rows:")
print(df.head())
print("\nMissing values:")
print(df.isnull().sum())
```

My Results:

```
INITIAL DATA OVERVIEW:
Dataset shape: (3276, 10)
Columns: ['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity', 'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability']

First 5 rows:
         ph    Hardness        Solids  Chloramines     Sulfate  Conductivity  \
0       NaN  204.890455  20791.318981     7.300212  368.516441    564.308654
1  3.716080  129.422921  18630.057858     6.635246         NaN    592.885359
2  8.099124  224.236259  19909.541732     9.275884         NaN    418.606213
3  8.316766  214.373394  22018.417441     8.059332  356.886136    363.266516
4  9.092223  181.101509  17978.986339     6.546600  310.135738    398.410813

   Organic_carbon  Trihalomethanes  Turbidity  Potability
0       10.379783        86.990970   2.963135           0
1       15.180013        56.329076   4.500656           0
2       16.868637        66.420093   3.055934           0
3       18.436524       100.341674   4.628771           0
4       11.558279        31.997993   4.075075           0

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

I discovered that three features had missing values that needed to be addressed during preprocessing.

3.2 Target Variable Analysis

I analyzed the target variable distribution to understand the classification problem:

```python
# Basic EDA for understanding
print("\nBASIC EDA:")
print("Target variable distribution:")
print(df['Potability'].value_counts())
print(f"Class balance: {df['Potability'].value_counts(normalize=True)}")
```

My Results:

```
BASIC EDA:
Target variable distribution:
Potability
0    1998
1    1278
Name: count, dtype: int64
Class balance: Potability
0    0.60989
1    0.39011
Name: proportion, dtype: float64
```

I found the dataset has a moderate class imbalance with approximately 61% non-potable and 39% potable water samples.

3.3 Feature Correlation Analysis

I investigated the relationships between predictor variables and the target:

```python
# Correlation analysis for feature engineering insights
correlation_matrix = df.corr()
print("\nTop correlations with target:")
target_corr = abs(correlation_matrix['Potability']).sort_values(ascending=False)
print(target_corr.head(6))
```

My Results:

```
Top correlations with target:
Potability         1.000000
Solids             0.033743
Organic_carbon     0.030001
Chloramines        0.023779
Sulfate            0.020476
Hardness           0.013837
Name: Potability, dtype: float64
```

I observed that all features show weak linear correlations with the target variable, suggesting that water potability determination involves complex, potentially non-linear interactions between multiple physicochemical parameters rather than simple dependence on individual variables.

---

4. Class Labelling for Target Variable

4.1 My Target Variable Classification Strategy

I assessed the target variable to determine the optimal classification approach:

```python
# Check if target variable is numerical or categorical
print(f"Target variable 'Potability' type: {df['Potability'].dtype}")
print(f"Unique values: {df['Potability'].unique()}")
print(f"Current distribution: {df['Potability'].value_counts().sort_index().to_dict()}")
```

My Results:

```
Target variable 'Potability' type: int64
Unique values: [0 1]
Current distribution: {0: 1998, 1: 1278}
```

My Decision: I found that the target variable is already categorical with a binary classification structure. Since the minority class represents 39% of observations (above the 0.30 threshold for acceptable balance), I decided to preserve the original binary classification scheme to maintain engineering relevance and regulatory compliance interpretation.

My Alternative Demonstration: To show my understanding of multi-class labelling, I created a 4-class pH-based categorization:

```
df['pH_quartiles'] = pd.qcut(df['ph'], q=4, labels=['Very_Acidic', 'Acidic', 'Basic', 'Very_Basic'])
df['pH_4class'] = pd.Categorical(df['pH_quartiles']).codes

print("pH-based 4-class distribution:")
ph_class_dist = df['pH_4class'].value_counts().sort_index()
print(ph_class_dist.to_dict())
print(f"Balanced distribution: {df['pH_4class'].value_counts(normalize=True).round(3).to_dict()}")
```

My Results:

```
pH-based 4-class distribution:
{0: 819, 1: 1065, 2: 573, 3: 819}
```

---

5. Feature Engineering and Feature Selection

5.1 My Data Preprocessing and Normalization

5.1.1 Missing Value Treatment

I handled missing values using median imputation:

```
numerical_cols = df.select_dtypes(include=[np.number]).columns
for col in numerical_cols:
    if df[col].isnull().sum() > 0:
        df[col].fillna(df[col].median(), inplace=True)
        print(f"Filled {col} missing values with median")
```

My Results:

```
DATA CLEANING:
Filled ph missing values with median
Filled Sulfate missing values with median
Filled Trihalomethanes missing values with median
Missing values after cleaning: 0
```

5.1.2 Feature Normalization

I applied StandardScaler normalization to ensure fair feature contribution:

```
# Normalization of numerical features
scaler = StandardScaler()
exclude_from_scaling = ['Potability', 'target_variable', 'turbidity_category', 'pH_quartiles',
                        'hardness_category', 'turbidity_cat_num', 'hardness_cat_num',
                        'ph_category', 'ph_cat_num', 'chloramines_category', 'chloramines_cat_num']
feature_cols = [col for col in df.columns if col not in exclude_from_scaling]
df_scaled = df.copy()
df_scaled[feature_cols] = scaler.fit_transform(df[feature_cols])
```

My Results:

```
Features normalized using StandardScaler:
Normalized features: ['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity', 'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'pH_4class']
```

5.2 My Integer Categorization of Features

I systematically converted continuous variables to categorical representations:

```
# Categorize Turbidity into 3 levels
df['turbidity_category'] = pd.cut(df['Turbidity'], bins=3, labels=['Low', 'Medium', 'High'])
df['turbidity_cat_num'] = pd.Categorical(df['turbidity_category']).codes

# Categorize Hardness into 4 levels
df['hardness_category'] = pd.cut(df['Hardness'], bins=4, labels=['Soft', 'Moderate', 'Hard', 'Very_Hard'])
df['hardness_cat_num'] = pd.Categorical(df['hardness_category']).codes

# Categorize pH into acid/neutral/basic levels
df['ph_category'] = pd.cut(df['ph'], bins=[0, 6.5, 7.5, 14], labels=['Acidic', 'Neutral', 'Basic'])
df['ph_cat_num'] = pd.Categorical(df['ph_category']).codes

# Categorize Chloramines levels
df['chloramines_category'] = pd.cut(df['Chloramines'], bins=3, labels=['Low', 'Medium', 'High'])
df['chloramines_cat_num'] = pd.Categorical(df['chloramines_category']).codes
```

My Results:

```
Integer categorization completed:
- Turbidity categories: turbidity_cat_num
0     554
1    2392
2     330
Name: count, dtype: int64
- Hardness categories: hardness_cat_num
0      41
1    1100
2    2001
3     134
Name: count, dtype: int64
- pH categories: {-1: 1, 0: 967, 1: 1249, 2: 1059}
- Chloramines categories: {0: 181, 1: 2671, 2: 424}
```

5.3 My Composite Feature Development

5.3.1 Ratio-Based Features

I created ratio-based features inspired by my understanding of water chemistry:

```
# Ratio features (based on correlation analysis)
df_scaled['ph_hardness_ratio'] = df_scaled['ph'] / (df_scaled['Hardness'] + 0.001)
df_scaled['solids_conductivity_ratio'] = df_scaled['Solids'] / (df_scaled['Conductivity'] + 0.001)
df_scaled['chloramines_sulfate_ratio'] = df_scaled['Chloramines'] / (df_scaled['Sulfate'] + 0.001)
```

5.3.2 Interaction Features

I developed interaction terms to capture synergistic effects:

```
# Product features (interaction effects)
df_scaled['organic_carbon_trihalomethanes_product'] = df_scaled['Organic_carbon'] * df_scaled['Trihalomethanes']
df_scaled['ph_conductivity_interaction'] = df_scaled['ph'] * df_scaled['Conductivity']
```

My Final Engineered Features:

1. ph_hardness_ratio
2. solids_conductivity_ratio
3. chloramines_sulfate_ratio
4. organic_carbon_trihalomethanes_product

5. ph_conductivity_interaction
6. turbidity_cat_num
7. hardness_cat_num
8. ph_cat_num
9. chloramines_cat_num

---

6. Decision Tree Model Development and Training

6.1 Model Architecture and Hyperparameter Configuration

A consistent decision tree architecture was implemented across all feature set evaluations to ensure fair comparative analysis:

```
# Train decision tree
dt = DecisionTreeClassifier(
    random_state=42,
    max_depth=10,
    min_samples_split=10,
    min_samples_leaf=5
)
```

Hyperparameter Justification:

- **max_depth=10:** Balances model complexity with interpretability
- **min_samples_split=10:** Prevents excessive partitioning of small subsets
- **min_samples_leaf=5:** Ensures statistical significance of leaf nodes

6.2 Feature Set Design and Experimental Framework

Six distinct feature sets were systematically designed to evaluate different aspects of feature engineering effectiveness:

```
# Updated feature_sets with more comprehensive sets
feature_sets = {
    'Set1_Original_9Features': original_features,
    'Set2_Top5Correlated': top_corr_features,
    'Set3_Original_Plus_Ratios': original_features + ['ph_hardness_ratio', 'solids_conductivity_ratio', 'chloramines_sulfate_ratio'],
    'Set4_Engineered_Features': ['ph_hardness_ratio', 'organic_carbon_trihalomethanes_product', 'turbidity_cat_num', 'hardness_cat_num', 'ph_conductivity_interaction'],
    'Set5_Best_Mixed': ['Sulfate', 'Conductivity', 'Organic_carbon', 'ph_hardness_ratio', 'organic_carbon_trihalomethanes_product', 'turbidity_cat_num'],
    'Set6_All_Categorical': ['turbidity_cat_num', 'hardness_cat_num', 'ph_cat_num', 'chloramines_cat_num']  # NEW SET
}
```

Experimental Design:

1. **Set1_Original_9Features:** Baseline performance using raw physicochemical parameters
2. **Set2_Top5Correlated:** Feature selection based on correlation magnitude
3. **Set3_Original_Plus_Ratios:** Integration of original and ratio-based features
4. **Set4_Engineered_Features:** Exclusive use of composite features
5. **Set5_Best_Mixed:** Curated combination of high-performing features

6. **Set6_All_Categorical:** Discretised categorical representations

6.3 Model Training and Validation Protocol

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X_subset, y, test_size=0.3, random_state=42, stratify=y
)
```

```
dt.fit(X_train, y_train)

# Make predictions
y_pred = dt.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

The stratified sampling approach ensures representative class distribution in both training and testing subsets, critical for unbiased performance estimation.

---

7. Comparative Performance Analysis

7.1 Quantitative Results Summary

Feature Set	Features (n)	Accuracy	Tree Depth	Leaves	Dominant Feature	Importance
Set3_Original_Plus_Ratios	12	0.6419	10	114	Sulfate	0.1623
Set1_Original_9Features	9	0.6338	10	125	Sulfate	0.1987
Set6_All_Categorical	4	0.6073	9	57	ph_cat_num	0.4643
Set5_Best_Mixed	6	0.5972	10	94	ph_hardness_ratio	0.2986
Set4_Engineered_Features	5	0.5921	10	95	ph_hardness_ratio	0.3293
Set2_Top5Correlated	5	0.5788	10	108	ph_hardness_ratio	0.3317

8. Summary of My Observations

Through my experimental analysis, I discovered several critical insights about feature engineering effectiveness in water potability classification:

My Key Findings:

1. Feature Integration Works Best: My Set3_Original_Plus_Ratios achieved the highest performance (64.19% accuracy), which taught me that combining domain-specific engineered features with original measurements enhances predictive capability. I achieved a 0.81 percentage point improvement over baseline features.

2. Categorical Features Are Surprisingly Effective: My Set6_All_Categorical achieved competitive performance (60.73% accuracy) using only 4 discretized features. This showed me that decision trees can effectively exploit categorical boundaries while reducing computational complexity.

3. Correlation-Based Selection Has Limitations: My Set2_Top5Correlated produced the lowest performance (57.88%), which taught me that simple correlation-based feature selection doesn't adequately capture the complex multivariate relationships essential for water potability determination.

4. Chemical Knowledge Matters: I consistently found Sulfate as the dominant decision factor across multiple feature sets, which aligns with what I learned about water quality standards in my coursework, validating that the model captures chemically relevant patterns.
5. Balance Between Complexity and Performance: My optimal model balanced predictive performance with reasonable interpretability (10 depth, 114 leaves), which I believe is crucial for engineering applications where I need to explain my decisions.

9. Appendix: Source Code Repository

My Source Code Access: I have made my complete Jupyter notebook implementation and associated data files available through the following shared repository:

https://github.com/S1zzX/COS40007-Artificial-Intelligence-for-Engineering-/tree/main/Portfolio_Assessment/Assessment1