**Documentation of Waste Classification**

By Ritu Shashula

**Dataset Overview:**

The dataset contains the following columns:

* **sensor\_id**: Identifier for the sensor collecting data.
* **timestamp**: Timestamp of data collection.
* **waste\_type**: Type of waste (e.g., organic, inorganic).
* **inductive\_property**: Inductive property value from the sensor.
* **capacitive\_property**: Capacitive property value from the sensor.
* **moisture\_property**: Moisture property value from the sensor.
* **infrared\_property**: Infrared property value from the sensor.

**Libraries Used:**

* **pandas, numpy**: Data loading and manipulation.
* **scikit-learn**: Machine learning models, preprocessing, evaluation metrics, and splitting.
* **matplotlib, seaborn**: Data visualization.

**Process:**

**1. Data Loading:**

* Dataset is uploaded from the user’s local machine using Google Colab.
* Initial dataset analysis is performed using .describe() and .info().

**2. Data Cleaning and Feature Engineering:**

* Missing values are handled by dropping rows with null values.
* New features:
  + **is\_organic**: Binary indicator for organic waste (1 = organic, 0 = inorganic).
* Dropped unnecessary columns: sensor\_id, timestamp, waste\_type.
* Feature scaling applied to numeric columns: inductive\_property, capacitive\_property, moisture\_property, and infrared\_property using StandardScaler.

**3. Exploratory Data Analysis:**

* **Pairplot**: Visualizes pairwise relationships between numerical features.
* **Heatmap**: Displays the correlation matrix.
* **Scatter Plot**: Examines the relationship between inductive and infrared properties.
* **Bar Plot**: Shows the distribution of organic vs. inorganic waste.

**4. Data Balancing:**

* Oversampled the minority class (organic waste) using resampling to balance the dataset.

**5. Train-Test Split:**

* Dataset is split into training and testing sets (80%-20%) with stratification.

**6. Dimensionality Reduction:**

* Applied PCA (Principal Component Analysis) to retain 95% of variance in features.

**7. Model Training and Evaluation:**

* Models used:
  + **Random Forest Classifier**
  + **Gradient Boosting Classifier**
  + **Support Vector Machine (SVM)**
* Accuracy of each model is computed and compared.
* Best model identified based on accuracy.

**8. Hyperparameter Tuning:**

* Randomized Search Cross-Validation (RandomizedSearchCV) applied to the best model for parameter optimization.

**9. Final Evaluation:**

* Final model tested on the test dataset.
* Classification report and accuracy score generated for performance evaluation.

**Metrics Computed:**

* **Accuracy**: Percentage of correct predictions.
* **Classification Report**: Precision, recall, F1-score, and support for each class.

**Visualization:**

* Pairplot of features.
* Heatmap showing feature correlations.
* Scatter plot: Inductive Property vs Infrared Property.
* Bar plot: Distribution of organic vs inorganic waste.

**Results:**

* Best Model: Identified dynamically based on accuracy.
* Final Accuracy: Displayed as a percentage.
* Detailed classification report provided.

**Code :**

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, RandomizedSearchCV

from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.decomposition import PCA

from sklearn.utils import resample

from google.colab import files

# Upload and Load Dataset

uploaded = files.upload()

file\_path = list(uploaded.keys())[0]

dataset = pd.read\_csv(file\_path)

print("\n--- Dataset Description ---\n")

print(dataset.describe())

print("\n--- Dataset Info ---\n")

print(dataset.info())

dataset.head()

# Handle Missing Values

print("\nChecking for missing values...")

print(dataset.isnull().sum())

dataset = dataset.dropna()

# Data Cleaning and Feature Engineering

dataset['timestamp'] = pd.to\_datetime(dataset['timestamp'])

dataset['is\_organic'] = dataset['waste\_type'].apply(lambda x: 1 if x == 'organic' else 0)

dataset\_cleaned = dataset.drop(columns=['sensor\_id', 'timestamp', 'waste\_type'])

# Feature Scaling

numeric\_columns = ['inductive\_property', 'capacitive\_property', 'moisture\_property', 'infrared\_property']

scaler = StandardScaler()

dataset\_cleaned[numeric\_columns] = scaler.fit\_transform(dataset\_cleaned[numeric\_columns])

# Pairplot

import seaborn as sns

import matplotlib.pyplot as plt

sns.pairplot(dataset)

plt.show()

# Heatmap of Correlation

import seaborn as sns

import matplotlib.pyplot as plt

import pandas as pd

# Filter out non-numeric columns

numeric\_dataset = dataset.select\_dtypes(include=['float64', 'int64'])

# Plot heatmap

plt.figure(figsize=(10, 6))

sns.heatmap(numeric\_dataset.corr(), annot=True, cmap='coolwarm')

plt.title('Correlation Heatmap')

plt.show()

# Correlation Matrix

dataset.hist(figsize=(10, 6))

plt.tight\_layout()

plt.show()

# Step 4: Scatter plot

plt.figure(figsize=(10, 5))

sns.scatterplot(data=dataset, x='inductive\_property', y='infrared\_property', hue='waste\_type')

plt.title('Scatter plot: Inductive Property vs Infrared Property')

plt.xlabel('Inductive Property')

plt.ylabel('Infrared Property')

plt.show()

# Bar plot

plt.figure(figsize=(10, 6))

sns.countplot(data=dataset, x='is\_organic', palette='Set1')

plt.title('Organic vs Non-Organic Waste')

plt.xlabel('Is Organic')

plt.ylabel('Count')

plt.show

# Balancing the Dataset

X = dataset\_cleaned.drop(columns=['is\_organic'])

y = dataset\_cleaned['is\_organic']

data = pd.concat([X, y], axis=1)

majority = data[data['is\_organic'] == 0]

minority = data[data['is\_organic'] == 1]

minority\_oversampled = resample(minority, replace=True, n\_samples=len(majority), random\_state=42)

data\_balanced = pd.concat([majority, minority\_oversampled]).sample(frac=1, random\_state=42).reset\_index(drop=True)

X\_balanced = data\_balanced.drop(columns=['is\_organic'])

y\_balanced = data\_balanced['is\_organic']

# Train-test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_balanced, y\_balanced, test\_size=0.2, random\_state=42, stratify=y\_balanced)

# Dimensionality Reduction

pca = PCA(n\_components=0.95, random\_state=42) # Retain 95% variance

X\_train\_pca = pca.fit\_transform(X\_train)

X\_test\_pca = pca.transform(X\_test)

# Model Training and Evaluation

models = {

'Random Forest': RandomForestClassifier(random\_state=42, n\_estimators=150, max\_depth=15),

'Gradient Boosting': GradientBoostingClassifier(random\_state=42, n\_estimators=150, learning\_rate=0.1, max\_depth=3),

'SVM': SVC(kernel='rbf', C=1, gamma='scale', random\_state=42)

}

best\_accuracy = 0

best\_model\_name = ""

for name, model in models.items():

model.fit(X\_train\_pca, y\_train)

y\_pred = model.predict(X\_test\_pca)

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"{name} Accuracy: {accuracy:.4f}")

if accuracy > best\_accuracy:

best\_accuracy = accuracy

best\_model\_name = name

# Hyperparameter Tuning for the Best Model

if best\_model\_name == 'Random Forest':

param\_grid = {

'n\_estimators': [100, 150, 200],

'max\_depth': [10, 15, 20],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

model = RandomizedSearchCV(RandomForestClassifier(random\_state=42), param\_grid, cv=3, scoring='accuracy', n\_jobs=-1, n\_iter=10, random\_state=42)

elif best\_model\_name == 'Gradient Boosting':

param\_grid = {

'n\_estimators': [100, 150, 200],

'learning\_rate': [0.05, 0.1, 0.2],

'max\_depth': [3, 5, 7]

}

model = RandomizedSearchCV(GradientBoostingClassifier(random\_state=42), param\_grid, cv=3, scoring='accuracy', n\_jobs=-1, n\_iter=10, random\_state=42)

elif best\_model\_name == 'SVM':

param\_grid = {

'C': [0.1, 1, 10],

'gamma': ['scale', 'auto'],

'kernel': ['rbf', 'poly']

}

model = RandomizedSearchCV(SVC(random\_state=42), param\_grid, cv=3, scoring='accuracy', n\_jobs=-1, n\_iter=10, random\_state=42)

model.fit(X\_train\_pca, y\_train)

print("Best Hyperparameters:", model.best\_params\_)

# Final Evaluation

y\_pred = model.best\_estimator\_.predict(X\_test\_pca)

final\_accuracy = accuracy\_score(y\_test, y\_pred)

report = classification\_report(y\_test, y\_pred)

print(f"Final Model ({best\_model\_name}) Accuracy: {final\_accuracy:.4f}")

print("Classification Report:\n", report)

print(final\_accuracy\*100)