

# ****Predicting Area Type (Residential or Industrial) Based on Air Quality Index (AQI) Using Artificial Neural Networks****

**Submitted by:** Rohan Kumar Jha (22112025)  
**Date:** 10/11/2024

## ****Introduction and Objectives****

The objective of this project is to predict the type of area (Residential or Industrial) based on air quality parameters using an **Artificial Neural Network (ANN)**. The dataset consists of various air quality indicators, including PM2.5, PM10, NOx, and other pollutants, along with the target variable indicating whether the area is **Residential** or **Industrial**. By training an ANN on this dataset, we can predict the area type based on air quality values, aiding in urban planning and environmental monitoring.

## ****Dataset Processing and Description****

### ****Loading the Dataset****

# Load the dataset

data = pd.read\_csv('C:/Users/asus/Downloads/Air-Quality-Index--AQI--main/Air-Quality-Index--AQI--main/city\_day.csv')

# Display basic information about the dataset to understand the structure

print(data.info())

print(data.head())

* **pd.read\_csv()**: This function reads the dataset from a CSV file and loads it into a DataFrame. The dataset contains various columns, including air quality indicators and demographics (Residential/Industrial).
* **data.info()**: This shows the structure of the DataFrame, listing all columns, their data types, and the number of non-null values.
* **data.head()**: Displays the first few rows of the dataset to help visualize the data.

### ****Data Preprocessing****

#### Handling Missing Data

# Drop rows where demographic data is missing

data = data.dropna(subset=['Demographics'])

* **dropna()**: This function is used to drop rows where the target variable Demographics is missing. It is crucial to only use rows with valid labels for training.

#### Defining Features and Target Variable

# Define features (pollutant columns) and target (demographics)

features = ['PM2.5', 'PM10', 'NO', 'NO2', 'NOx', 'NH3', 'CO', 'SO2', 'O3', 'Benzene', 'Toluene', 'Xylene']

X = data[features]

y = data['Demographics']

* **Features (X)**: The selected columns represent air quality parameters such as PM2.5, NO2, CO, and others.
* **Target (y)**: The Demographics column, which indicates whether the area is Residential or Industrial, is used as the target for classification.

#### Imputing Missing Values in Features

# Impute missing values in the features with the mean

imputer = SimpleImputer(strategy='mean')

X = imputer.fit\_transform(X)

* **SimpleImputer(strategy='mean')**: Replaces missing values in the features with the mean of the respective columns. This ensures that no missing data is present in the feature set.

#### Label Encoding the Target Variable

# Encode the categorical target variable (Demographics)

label\_encoder = LabelEncoder()

y\_encoded = label\_encoder.fit\_transform(y)

* **LabelEncoder()**: Converts the categorical labels (Residential, Industrial) into numerical values (0 for Residential, 1 for Industrial), as machine learning models require numeric data.

#### Feature Scaling

# Standardize the features for better model performance

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

* **StandardScaler()**: Scales the features so that they have a mean of 0 and a standard deviation of 1. Feature scaling is important for ensuring that the model learns effectively, especially for ANN models.

#### Splitting the Data into Training and Testing Sets

# Split the dataset into training and testing sets

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

* **train\_test\_split()**: Splits the dataset into 80% training data and 20% testing data. This ensures that the model is evaluated on unseen data.

## ****Model Description****

### ****Artificial Neural Network (ANN) Architecture****

The model architecture consists of:

1. **Input Layer**: The number of neurons in this layer equals the number of features (12 in this case).
2. **Hidden Layers**: Two hidden layers with 64 and 32 neurons, using the ReLU (Rectified Linear Unit) activation function.
3. **Output Layer**: The output layer consists of 2 neurons (for Residential and Industrial classes) and uses the **Softmax** activation function.

model = Sequential()

model.add(Dense(64, input\_shape=(X\_train.shape[1],), activation='relu')) # First hidden layer

model.add(Dense(32, activation='relu')) # Second hidden layer

model.add(Dense(16, activation='relu')) # Third hidden layer (optional)

model.add(Dense(len(np.unique(y\_encoded)), activation='softmax')) # Output layer with softmax activation

* **Input Layer**: The input\_shape is the number of features, which is 12.
* **Hidden Layers**: Two hidden layers, each with ReLU activation, which helps in learning complex patterns.
* **Output Layer**: The output layer has two neurons representing the two classes (Residential and Industrial). Softmax activation converts the output into probabilities, and the class with the highest probability is selected.

### ****Why Softmax?****

Softmax is used in the output layer because it converts raw output scores into probabilities for each class. It is suitable for multi-class classification, as it normalizes the scores so they sum to 1.

## ****Model Implementation, Training, and Optimization****

### ****Model Compilation and Training****

# Compile the model with a suitable loss function and optimizer for classification

model.compile(optimizer=Adam(learning\_rate=0.001), loss='sparse\_categorical\_crossentropy', metrics=['accuracy'])

# Train the model

history = model.fit(X\_train, y\_train, epochs=50, batch\_size=16, validation\_split=0.2, verbose=1)

* **model.compile()**: The model is compiled with:
  + **Adam optimizer**: Adaptive learning rate optimizer suitable for training deep learning models.
  + **Loss function**: **Sparse categorical cross-entropy** is used because the target variable is integer-encoded.
  + **Metrics**: Accuracy is used to measure model performance.
* **Training the Model**: The model is trained for **50 epochs** using a **batch size of 16**. The validation split is set to 0.2, meaning 20% of the training data will be used for validation. The verbose=1 parameter ensures that training progress is shown.

### ****Model Evaluation and Prediction****

# Predict on the test set

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

y\_pred\_classes = np.argmax(y\_pred, axis=1) # Convert predicted probabilities to class labels

# Display classification report and accuracy score

print("Classification Report:\n", classification\_report(y\_test, y\_pred\_classes, target\_names=label\_encoder.classes\_))

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_classes)}")

output-

185/185 [==============================] - 0s 1ms/step

Classification Report:

precision recall f1-score support

industrial 0.53 0.59 0.56 1893

residential 0.80 0.75 0.77 4014

accuracy 0.70 5907

macro avg 0.66 0.67 0.67 5907

weighted avg 0.71 0.70 0.70 5907

Accuracy: 0.7003555104113763

* **Prediction**: After training, the model is used to make predictions on the test set. **np.argmax(y\_pred, axis=1)** converts the predicted probabilities into class labels (0 for Residential, 1 for Industrial).
* **Evaluation**: A classification report is generated, which includes metrics such as precision, recall, F1-score for each class. The accuracy of the model is also printed.

## ****Results and Evaluation****

### ****Sample Predictions****

We can test the model with sample data to verify the predictions.

# Sample 1 - Residential

sample\_data = [54.73, 94.12, 3.49, 12.79, 9.73, 22.79, 0.58, 8.21, 30.21, 0.08, 2.23, 0.15]

sample\_data\_scaled = scaler.transform([sample\_data])

pred = model.predict(sample\_data\_scaled)

print(label\_encoder.inverse\_transform(np.argmax(pred, axis=1)))

output-

1/1 [==============================] - 0s 19ms/step

Predicted Demographic: industrial

c:\Users\asus\AppData\Local\Programs\Python\Python311\Lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but StandardScaler was fitted without feature names

warnings.warn(

* **Sample Prediction**: A sample input representing air quality values is passed to the model. After scaling the features using the same scaler as during training, the model predicts whether the area is **Residential** or **Industrial**.

### ****Detailed Code Explanation for Sample Predictions****

* For each sample input, the model first scales the features, then makes a prediction using **model.predict()**. The prediction probabilities are converted to class labels using **np.argmax()**, and the result is printed using **label\_encoder.inverse\_transform()** to convert the numeric prediction back to the original class label.