MACHINE LEARNING METHODS APPLIED IN AIR QUALITY PREDICTION

Mihai-Claudiu Vieru [1]

Mădălina Cărbureanu [2]

[1] Romania
[2] Petroleum-Gas University of Ploiesti, Romania
email: mcarbureanu@upg-ploiesti.ro

DOI: 10.51865/JPGT.2024.01.01

ABSTRACT

Air quality is an important environmental component that has a significant influence on public health and well-being. Poor air quality can cause a variety of health problems, including respiratory and cardiovascular disorders. Therefore, there is a growing demand for air quality prediction tools to enable consumers and authorities to take the best decisions and to implement the necessary actions to reduce air pollution. The present paper describes an innovative application that uses machine learning techniques to supply to the users real-time air quality predictions made on past data from their unique location. The Scikit-learn Python package was used to implement five machine learning algorithms, including K-Nearest Neighbors, Random Forest, Gradient Boosting, Support Vector Regression (SVR) and AdaBoost. To achieve robust model performance, compatibility with cross-validation approaches was evaluated. The obtained results indicate that these machine learning techniques are successful at forecasting air quality. The AdaBoost method emerged as the best accurate predictor after extensive investigation, closely followed by Gradient Boosting, SVR, Random Forest, and K-Nearest Neighbors. Furthermore, the investigation also focused on the adapted handling of inaccurate data and providing graphical visualizations to highlight the algorithm's efficacy.

Keywords: Python, K-Nearest Neighbors, SVR, AdaBoost, Random Forest, Gradient Boosting, Machine Learning, air quality, cross-validation

INTRODUCTION

Air quality is undeniably a critical environmental component which has a strong impact on public health and overall well-being. The air quality significantly impacts people everyday life, since poor air quality can lead to a multitude of health problems, ranging from respiratory ailments to cardiovascular disorders. With growing concerns regarding air pollution, there has been an unprecedented surge in demand for reliable and real-time tools for real time information of the population regarding the quality of the air they breathe. Such information is vital not only for empowering citizens to make informed decisions about their outdoor activities but also for aiding authorities in developing and implementing effective strategies [12, 15, 17, 19].

In recent years, the machine learning (ML) methods has experienced ground-breaking advancements, opening exciting new avenues for forecasting and monitoring air quality. This convergence of technology and environmental science has opened a new era where ML approaches can harness vast volumes of data from diverse sources, including an extensive network of sensors and high-resolution satellite imagery. These innovations are essential in developing highly accurate air quality predictions that transcend traditional models, providing valuable insights into air quality dynamics within specific regions [16].

The paper presents an innovative application that harnesses the power of ML in order to supply to the user's real-time air quality predictions. By leveraging historical data, combined with advanced algorithms, this application brings a high level of precision to air quality monitoring. It not only empowers individuals to make informed decision about their daily activities but also equips local authorities with the tools they need to proactively address air quality concerns and protect the health and well-being of their communities.

The quality of air depends on the composition and cleanliness of the atmosphere, a critical aspect of environmental health and sustainability. According [12, 17, 18], the key parameters routinely monitored to assess air quality are: particulate matter (PM2.5 and PM10) (these measurements track the concentration of fine particles suspended in the air, which can have adverse effects on respiratory health), ground-level Ozone (O3) (the monitoring of ozone levels is crucial, as high concentrations near the Earth's surface can cause respiratory problems and harm vegetation), nitrogen dioxide (NO_2) and sulphur dioxide (SO_2) (these gases are emitted from combustion processes and industrial activities and can lead to respiratory and environmental problems when present in high amounts), carbon monoxide (CO) (the monitoring of CO levels is essential as this colourless, odourless gas can be lethal in high concentrations), volatile organic compounds (VOCs) **(**VOCs are organic chemicals that can vaporize into the air and contribute to air pollution; they play a role in the formation of ground-level ozone and smog) and carbon dioxide (CO_2) (while not a pollutant itself, monitoring CO_2 levels is vital to assess greenhouse gas emissions and their contribution to climate change). Air quality standards and regulations vary by region and country, but they typically adhere to guidelines established by organizations such as the World Health Organization (WHO) and the Environmental Protection Agency (EPA) in the United States [20, 22]. These standards set the permissible concentration levels for various air pollutants to protect human health and the environment. Maintaining and improving air quality is crucial to ensure the well-being of both human populations and ecosystems, making it a paramount concern in environmental and public health policies worldwide. In this context, companies like IQAir and Airly play an important roles in enhancing awareness and enabling individuals to actively engage in safeguarding air quality. They achieve this through their innovative software interfaces. IQAir (Figure 1), for instance, provides a user-friendly mobile application that empowers users to monitor, manage and optimize indoor air quality. Its real-time notifications and intuitive controls enable individuals to take proactive steps towards healthier living environments [16]. On the other hand, Airly (Figure 2) offers a web and mobile application interface that displays real-time air quality data on a map. This dynamic platform equips users with the tools to access detailed information on pollutant levels, historical trends and alerts, thereby fostering informed decisions and community engagement to address outdoor air quality concerns [13]. The World Air Quality Index (WAQI) and AirNow are two important applications that help to monitor

and improve air quality. A well-known program called WAQI (World Air Quality Index) gathers data on air quality from monitoring stations across the world and offers real-time information on pollutants through a user-friendly mobile app and website (Figure 3) [14, 23]. The Environmental Protection Agency (EPA) of the United States provides AirNow (Figure 4), which allows users to plan their daily activities and make air quality-related decisions, with real-time data on the country's air quality, including the Air Quality Index (AQI), historical data, forecasts, and pollution maps [14, 21, 22].

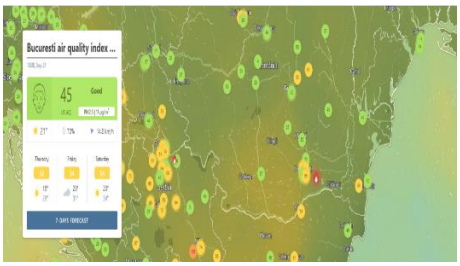

Figure 1. *IQAir user Interface [16]*

Figure 2. *Airly user interface [13]*

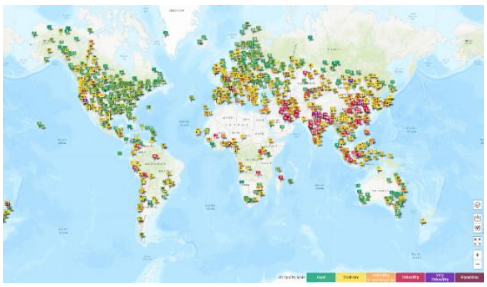

Figure 3. *WAQI user Interface [23]*

Figure 4. *AirNowuser Interface [14]*

THE AIR QUALITY PREDICTION APPLICATION DEVELOPMENT

The developed application AirPredict is a cutting-edge solution designed to offer users real-time information about air quality in their surroundings. The application employs ML algorithms to analyse data from a range of sources, including air quality monitoring stations, weather data and existing databases. AirPredict uses a series of pollutants that have been registered in the past months or years, to determine and predict the AQI (Air Quality Index) for the following day. The pollutants that are measured are SO_2, NO_2, O3, CO, C6H6, PM10, PM2.5, Pb, As, Cd, Ni, Benzo(a)pyrene (Table 1). Using these pollutants, AirPredict can determine the AQI factor on which the following day prediction is based on [12, 16, 17, 19].

Table 1. *Pollutant's Sub-Index formula* [1]

Pollutant	**Sub-Index Formula**
SO_2 (Sulfur Dioxide)	SO_2 Sub-Index = IHI * (C - ILo) / (IHi - ILo) + 0
NO_2 (Nitrogen Dioxide)	NO_2 Sub-Index = IHI * (C - ILo) / (IHi - ILo) + 0
O3 (Ground-level Ozone)	O3 Sub-Index = IHI * (C - ILo) / (IHi - ILo) + 0
CO (Carbon Monoxide)	CO Sub-Index = IHI * (C - ILo) / (IHi - ILo) + 0
C6H6 (Benzene)	C6H6 Sub-Index = IHI * (C - ILo) / (IHi - ILo) + 0

PM10 (Particulate Matter ≤ 10µm)	PM10 Sub-Index = IHI * (C - ILo) / (IHi - ILo) + 0
PM2.5 (Particulate Matter ≤ 2.5µm)	PM2.5 Sub-Index = IHI * (C - ILo) / (IHi - ILo) + 0
Pb (Lead)	Pb Sub-Index = IHI * (C - ILo) / (IHi - ILo) + 0
Arsenic	Arsenic Sub-Index = IHI * (C - ILo) / (IHi - ILo) + 0
Cadmium	Cadmium Sub-Index = IHI * (C - ILo) / (IHi - ILo) + 0
Nickel	Nickel Sub-Index = IHI * (C - ILo) / (IHi - ILo) + 0
Benzo(a)pyrene	B(a)p Sub-Index = IHI * (C - ILo) / (IHi - ILo) + 0

In table 1, IHI is the pollutant index, C is the pollutant concentration, ILo is the pollutant lower concentration breakpoint while IHi represents the upper concentration breakpoint for the air pollutant. The AQI is calculated by taking the highest individual pollutant index and rounding it to the nearest integer.

The developed application AirPredict has carried out an in-depth analysis using a dataset extracted from Kaggle, comprising fifteen thousand data points for one point five years [17]. After meticulously cleaning and filtering the data to remove irregularities, a refined dataset of twelve thousands entries remained as it can be observed in table 2.

Table 2. *The AirPredict dataset (selection)* [17]

Date	**01-01-2022**	**20-01-2022**	**25-01-2022**	**28-02-2022**
Location	Bucharest	Brasov	Galati	Ploiesti
AQI (Air Quality Index)	52.52	44.94	56.41	40.25
SO_2 (Sulfur Dioxide)	196.76	313.8	196.42	240.65
NO_2 (Nitrogen Dioxide)	87.03	117.75	68.9	176.88
O3 (Ozone)	69.82	71.24	6.07	96.8
CO (Carbon Monoxide)	0.73	3.7	5.56	2.61
C6H6 (Benzene)	3.58	2.31	3.84	0.78
PM10 (Particulate Matter 10)	0.99	20.08	38.61	27.66
PM2.5 (Particulate Matter 2.5)	6.51	10.11	13.65	3.3
Pb (Lead)	0.22	0.29	0.33	0.48
Arsen	1.85	2.59	5.05	4.1
Cadmiu	4.9	2.82	4.67	1.6
Nichel	5.56	3.3	4.47	16.79
Benzo(a)pyrene	0.41	0.98	0.37	0.97

A substantial dataset is crucial for precise predictions, especially when forecasting over an extended period. However, it's important to note that the fifteen thousand data points are spread across thirty cities from Romania, resulting in an average of about four hundred data points per city. This city-specific distribution plays an important role in the prediction accuracy. It is known the fact that ML algorithms excel when provided with huge amount of data (in this case data from specific region). This approach not only offers a comprehensive view of air quality within each city but also enhances the algorithms' ability to identify the local nuances and trends. Consequently, predictions of the Air Quality Index (AQI) achieve higher accuracy when grounded in a wealth of data extracted

from specific areas [16]. The materials and methods used to create AirPredict application uses ML techniques to provide real-time air quality predictions based on past data from a given location. The application uses the capabilities of five different ML algorithms to build an accurate air quality prediction model, such as:

- **K-Nearest Neighbors (K-NN):** is a classification technique based on the principle of data similarity, with predictions made by considering the majority class of the k-nearest neighbors; it excels in capturing localized patterns in air quality data [5, 7, 10, 11]; the developed source code for K-NN algorithm is presented in Figure 5;
- **Random Forest:** is an ensemble learning method that excels with complex datasets; it combines multiple decision trees to provide reliable predictions while mitigating overfitting, adept at capturing intricate relationships in air quality data [5, 8, 10]; the developed source code for Random Forest is presented in Figure 6;
- **Gradient Boosting:** it iteratively improves predictive models by correcting errors of previous iterations; it refines air quality predictions by capturing both global and local data patterns, valuable for long-term forecasting [5, 7, 6, 11]; the developed source code for Gradient Boosting algorithm is presented in Figure 7;
- **Support Vector Regression (SVR):** is a variation of Support Vector Machines (SVM), creates precise regression models by finding an optimal hyperplane while minimizing the difference between predicted and actual values [5, 10, 11]; the developed source code for SVR is presented in Figure 8;
- **AdaBoost (Adaptive Boosting):** it enhances prediction accuracy by combining multiple weak learners and assigning higher weight to misclassified data points [5, 10, 11]; the developed source code for AdaBoost algorithm (Figure 10) is presented in Figure 9.

```
5
6   # Load the dataset from the CSV file
7   data = pd.read_csv('C:/Users/X/Desktop/AirPredict/aqi2.csv')
8
9   # Split the dataset into features (X) and target (Y)
10  X = data.drop(['Date', 'Location', 'AQI'], axis=1)
11  y = data['AQI']
12
13  # Split the data into training and testing sets
14  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
15
16  # Create the K-Neighbors regression model
17  model = KNeighborsRegressor()
18
19  # Fit the model to the training data
20  model.fit(X_train, y_train)
21
22  # Predict AQI for the test data
23  y_pred = model.predict(X_test)
24
25  # Calculate evaluation metrics
26  explained_variance = explained_variance_score(y_test, y_pred)
27  mean_absolute_err = mean_absolute_error(y_test, y_pred)
28  mean_absolute_percentage_err = mean_absolute_percentage_error(y_test, y_pred)
29  median_absolute_err = median_absolute_error(y_test, y_pred)
```

Figure 5. *K-NN source code*

```
5
6   # Load the dataset from the CSV file
7   data = pd.read_csv('C:/Users/X/Desktop/AirPredict/aqi2.csv')
8
9   # Split the dataset into features (X) and target (Y)
10  X = data.drop(['Date', 'Location', 'AQI'], axis=1)
11  y = data['AQI']
12
13  # Split the data into training and testing sets
14  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
15
16  # Create the Random Forest model
17  model = RandomForestRegressor()
18
19  # Fit the model to the training data
20  model.fit(X_train, y_train)
21
22  # Predict AQI for the test data
23  y_pred = model.predict(X_test)
24
25  # Calculate evaluation metrics
26  explained_variance = explained_variance_score(y_test, y_pred)
27  mean_absolute_err = mean_absolute_error(y_test, y_pred)
28  mean_absolute_percentage_err = mean_absolute_percentage_error(y_test, y_pred)
29  median_absolute_err = median_absolute_error(y_test, y_pred)
```

Figure 6. *Random Forest source code*

```

# Load the dataset from the CSV file
data = pd.read_csv('C:/Users/X/Desktop/AirPredictD/aqi2.csv')

# Split the dataset into features (X) and target (Y)
X = data.drop(['Date', 'Location', 'AQI'], axis=1)
y = data['AQI']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the Gradient Boosting regression model
model = GradientBoostingRegressor(random_state=42)

# Fit the model to the training data
model.fit(X_train, y_train)

# Predict AQI for the test data
y_pred = model.predict(X_test)

# Calculate evaluation metrics
explained_variance = explained_variance_score(y_test, y_pred)
mean_absolute_err = mean_absolute_error(y_test, y_pred)
mean_absolute_percentage_err = mean_absolute_percentage_error(y_test, y_pred)
median_absolute_err = median_absolute_error(y_test, y_pred)
```

Figure 7. *Gradient Boosting source code*

```

# Load the dataset from the CSV file
data = pd.read_csv('C:/Users/X/Desktop/AirPredict/aqi2.csv')

# Split the dataset into features (X) and target (Y)
X = data.drop(['Date', 'Location', 'AQI'], axis=1)
y = data['AQI']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42

# Create the SVR model
model = SVR()

# Fit the model to the training data
model.fit(X_train, y_train)

# Predict AQI for the test data
y_pred = model.predict(X_test)

# Calculate evaluation metrics
explained_variance = explained_variance_score(y_test, y_pred)
mean_absolute_err = mean_absolute_error(y_test, y_pred)
mean_absolute_percentage_err = mean_absolute_percentage_error(y_test, y_pred)
median_absolute_err = median_absolute_error(y_test, y_pred)
```

Figure 8. *SVR source code*

```

# Load the dataset from the CSV file
data = pd.read_csv('C:/Users/X/Desktop/AirPredict/aqi2.csv')

# Split the dataset into features (X) and target (Y)
X = data.drop(['Date', 'Location', 'AQI'], axis=1)
y = data['AQI']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create the AdaBoost regression model
model = AdaBoostRegressor(random_state=42)

# Fit the model to the training data
model.fit(X_train, y_train)

# Predict AQI for the test data
y_pred = model.predict(X_test)

# Calculate evaluation metrics
explained_variance = explained_variance_score(y_test, y_pred)
mean_absolute_err = mean_absolute_error(y_test, y_pred)
mean_absolute_percentage_err = mean_absolute_percentage_error(y_test, y_pred)
median_absolute_err = median_absolute_error(y_test, y_pred)
```

Figure 9. *AdaBoost source code*

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier

class AdaBoostClassifier:
    def __init__(self, n_estimators=50):
        self.n_estimators = n_estimators
        self.alphas = []
        self.models = []

    def fit(self, X, y):
        n_samples, n_features = X.shape
        w = np.full(n_samples, 1 / n_samples)  # Initialize sample weights

        for _ in range(self.n_estimators):
            model = DecisionTreeClassifier(max_depth=1)
            model.fit(X, y, sample_weight=w)
            y_pred = model.predict(X)

            # Calculate weighted error
            err = np.sum(w * (y_pred != y)) / np.sum(w)

            # Calculate alpha (classifier weight)
            alpha = 0.5 * np.log((1 - err) / max(err, 1e-10))
            self.alphas.append(alpha)

            # Update sample weights
            w = w * np.exp(-alpha * y * y_pred)
            w /= np.sum(w)  # Normalize weights

            self.models.append(model)

    def predict(self, X):
        n_samples = X.shape[0]
        preds = np.zeros(n_samples)

        for alpha, model in zip(self.alphas, self.models):
            preds += alpha * model.predict(X)

        return np.sign(preds)

# Example usage:
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X, y = make_classification(n_samples=100, n_features=20, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

ada_boost = AdaBoostClassifier(n_estimators=50)
ada_boost.fit(X_train, y_train)
y_pred = ada_boost.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

Figure 10. *AdaBoost algorithm code*

For the algorithms development and training was used Python's Scikit-Learn module due to its extensive toolkit, simplicity, and rich documentation [5, 6, 9]. Also, for assessing the studied ML algorithms (K-NN, Gradient boosting, Random Forest, SVR and AdaBoost's) durability and generalization was used cross-validation method presented in Figure 11. It involves iteratively training and testing of the models on different dataset subsets to evaluate predictive performance across diverse data circumstances [2, 3, 4, 5, 10, 11].

```
# Evaluation of every algorithm using Cross-Validation
def evaluate_algorithm(name, model, metric_scorer):
    scores = cross_val_score(model, X_scaled, y, cv=5, scoring=metric_scorer)
    return scores

for name, model in algorithms.items():
    for metric_name, metric_scorer in metrics.items():
        # Perform parallel evaluation and calculate the metric scores
        scores = Parallel(n_jobs=-1)(delayed(evaluate_algorithm)(name, model, metric_scorer) for _ in range(5))
        metric_scores = [score.mean() for score in scores]
        results[metric_name][name] = metric_scores
```

Figure 11. *Cross-Validation method for ML algorithms*

In Figure 12, is presented the developed application AirPredict interface which provides to the user the following options:

- The selection of the desired ML algorithm **(**Random Forest, Ada Boost, Gradient Boosting, K-NN or SVR**)** to generate air quality predictions; the algorithms utilize historical data to develop predictive models (Table 2);
- **Graph** option for visual representations (graphs option) of different aspects of the prediction process; additional graphs, such as those illustrating R-squared (R2) or Mean Absolute Error (MAE), may be included to assess model accuracy;
- **Individual graph** option used to obtain specific graphs generated for each used ML algorithm; these graphs display various performance metrics, errors or other relevant information;
- The **Compare** option initiates the execution of all studied ML algorithms, followed by a rigorous evaluation process utilizing the cross-validation methodology. The goal is to identify the optimal ML algorithm from the analyzed ones and to provide a nuanced assessment of their relative performances in terms of prediction accuracy and generalization capabilities.

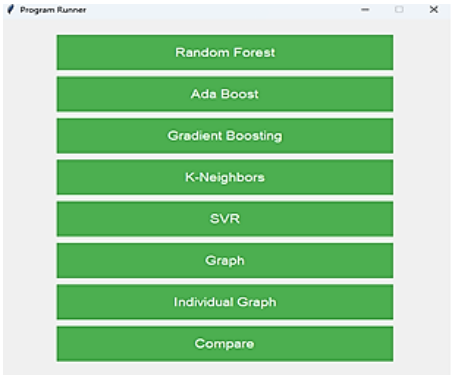

Figure 12*. AirPredict user interface*

So, cross-validation method was used to evaluate (using the parameters presented in Table 3) how well the K-NN, Gradient boosting, Random forest, SVR, and AdaBoost generalize the unknown data. It's a crucial step to ensure that the developed models are robust and reliable [2].

Table 3. *Cross-validation parameters [2]*

Parameter	Description
Cross-validation	A resampling technique used to assess model performance by partitioning the dataset into training and validation subsets.
K-fold cross-validation	Divides the dataset into *k* subsets (folds) for training and validation; iteratively, one-fold is used for validation while the remaining *k-1* folds are used for training; this process repeats 'K' times, and performance metrics are averaged.
Stratified k-fold cross-validation	Ensures that each fold has a similar distribution of target classes as the original dataset, suitable for imbalanced datasets.
Leave-one-out cross-validation (LOOCV)	A special case of *k*-fold cross-validation where *k* equals the number of data points, resulting in *n* iterations; each data point serves as the validation set while the rest are used for training.
Shuffle-split cross-validation	Randomly shuffles the data and splits it into training and validation sets for *n_splits* iterations; it allows control over the number of splits and validation set size.
Performance metrics	Evaluation metrics such as Mean Absolute Error (MAE), Root Mean Square Error (RMSE), R-squared (R2), and others are used to assess algorithm performance.
Best algorithm selection	The algorithm with the lowest error metrics (e.g., MAE, RMSE) or highest goodness-of-fit (e.g., R2) scores across cross-validation iterations is typically considered the best performer.
Generalization assessment	Cross-validation helps determine how well a model generalizes to unknown data; lower variability in performance across folds suggests better generalization.
Hyper parameter tuning	Cross-validation is used to fine-tune hyper parameters of machine learning algorithms by assessing their impact on model performance across different folds.
Overfitting detection	High variance in performance across folds may indicate overfitting, where the model fits the training data too closely and fails to generalize; cross-validation helps detect this issue.
Bias-variance trade-off	Cross-validation aids in understanding the trade-off between model bias (under fitting) and variance (overfitting) by examining how different algorithms strike this balance.

Based on the results obtained through cross-validation the application determines the best-performing ML algorithm. A rational explanation for the choice made is provided using the following evaluation metrics (Table 4):

- **R2** (**R-squared**) is a statistical measure that quantifies the proportion of the variance in the dependent variable (e.g., AQI) that is explained by the independent variables (e.g., predictors used in a model); it ranges from 0 to 1, where 0 indicates that the model explains none of the variance and 1 indicates a perfect fit; higher R2 values suggest that the model's predictions closely match the actual values, indicating a better fit [2];

- **MAE** (**Mean Absolute Error**) calculates the average absolute difference between predicted and actual values for a set of data points; it provides a measure of the model's accuracy in predicting the actual AQI values; lower MAE values indicate that the model's predictions are closer to the actual values, reflecting higher accuracy [2];
- **MAPE (Mean Absolute Percentage Error**) measures the average absolute percentage difference between predicted and actual values; it quantifies the relative accuracy of the model's predictions, making it suitable for assessing performance when the scale of the data varies; lower MAPE values indicate that the model's percentage errors are smaller, implying better relative accuracy [2];
- **MedAE** (**Median Absolute Error**) computes the median (middle value) of the absolute differences between predicted and actual values; it offers a robust measure of central tendency for the prediction errors and is less sensitive to outliers than MAE; MedAE is particularly useful when the dataset contains extreme values or outliers [2].

Table 4. *Evaluation metrics mathematical formulas [4]*

Metric	Formula
R2 Score	$1 - \frac{SumofSquaresofResiduals}{TotalSumofSquares}$
MAE (Mean Absolute Error)	$\frac{1}{n}\sum_{i=1}^{n}\lvert Y_i - y_i\rvert$
MAPE (Mean Absolute Percentage Error)	$\frac{1}{n}\sum_{i=1}^{n}\left\lvert\frac{Y_i - y_i}{y_i}\right\rvert * 100\%$
Median Absolute Error (MedAE)	$(\lvert Y_i - y_i\rvert)$

In table 4, n represents the number of data points, Y_i represents the actual AQI value for data point I, while y_i represents the predicted AQI value for data point i.

RESULTS AND DISCUSSIONS

To develop a reliable air quality prediction system, the application undertook a thorough review of historical data from thirty cities in Romania, a selection of data being presented in the Table 2 [17]. The primary goal was to evaluate the efficacy of five different ML algorithms in terms of generating real-time air quality predictions, efficiency measured following the next criteria [2, 10, 11]:

- **Algorithms performance**: the research provided useful insights into the effectiveness of used ML techniques; it is important to highlight that all algorithms performed well in processing the prior year's datasets, demonstrating their utility in air quality prediction [4, 5];
- **Managing aberrant data**: one major accomplishment was the algorithm's ability to successfully handle erroneous data; in environmental datasets, outliers and unexpected data points are presented and dealing with them is critical for good forecasts [4, 5, 7, 11];

- **Method of cross-validation**: the application uses the cross-validation approach to guarantee the resilience and generalization of the K-NN, Gradient boosting, Random Forest, SVR and AdaBoost models; was tested how well the models would perform on unknown data using this method; the cross-validation findings, together with individual performance measures, offered useful insights into the ranking of ML algorithms [5, 7].

Table 5 presents a comprehensive overview of the evaluation metrics used to assess the performance of the analysed ML algorithms.

Table 5. *Evaluation metrics for the analyzed ML algorithms*

Algorithm	Score R2	MAE	MAPE	MedAE
SVR	0.00262141 95429361	6.64047789 7958316	0.14030579 457355047	5.76661036 767187
Random Forests	0.03187172 49386939	6.72374243 6412316	0.14214603 048160224	5.86944999 9999979
AdaBoost	**0.00161907 78546933**	**6.63617991 32395385**	**0.14014311 785641947**	**5.72643407 8002875**
Gradient Boosting	0.01153712 50445714	6.67209757 12301085	0.14107339 578521325	5.73340484 9539017
K-NN	0.22449367 38071974	7.37010174 0294512	0.15482715 428068558	6.46500000 0000003

From table 5, the following information can be obtained:

- The top-performing algorithm was AdaBoost (with the learning curve graph from Figure 13) based on superior evaluation metrics; its adaptability and concentration on misclassified data points contributed to its high accuracy in assessing air quality levels; AdaBoost is better than the other algorithms based on the metrics shown in table 5 because it has a higher R2 score, a lower MAE and MAPE, and a lower MedAE; this means that AdaBoost is better at predicting the target variable accurately and with low error;
- In terms of performance, Gradient Boosting (with the learning curve graph from Figure 14) closely followed AdaBoost; this ensemble learning technique performed well in terms of improving predictions and minimizing mistakes, making it a good contender for estimating air quality;
- SVR (with the learning curve graph from Figure 15) performed well, producing solid regression models for air quality prediction; its capacity to discover the best-fitting hyperplane while reducing prediction errors was demonstrated to be useful;
- Random Forest (with the learning curve graph from Figure 16) gained a decent position in the ranking thanks to its stability and ability to handle complicated datasets; it contributed to reliable air quality assessments; however, it lagged somewhat behind the best algorithms;
- While K-NN (with the learning curve graph from Figure 17) performed well, it was near the bottom of achieved ranking; while it was effective in some cases, its emphasis on proximity-based categorization appears to be less suited to the complexities of air quality prediction.

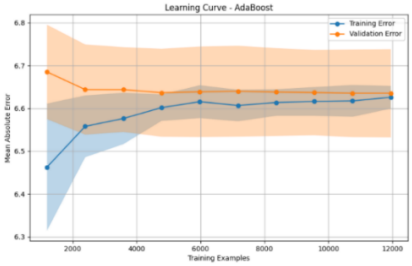

Figure 13. *AdaBoost learning curve*

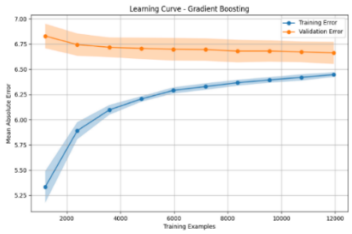

Figure 14. *Gradient Boosting learning curve*

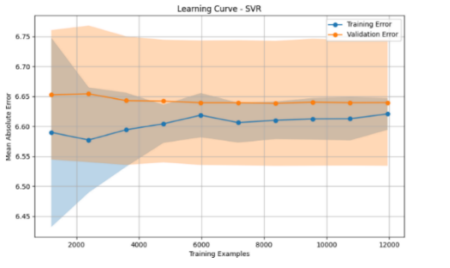

Figure 15. *SVR learning curve*

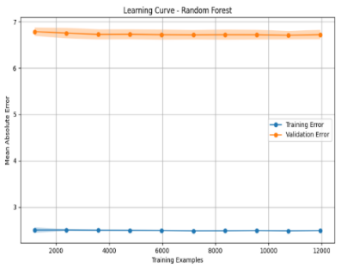

Figure 16. *Random Forest learning curve*

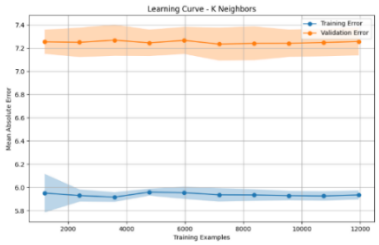

Figure 17. *K-NN learning curve*

The obtained learning curve presented above, illustrates that initially, the model quickly learns from training data, resulting in a lower training error than the validation error. However, over time, the model starts overfitting the training data, causing the training error to decrease while the validation error increases. The initial drop in the validation error from 6.645 suggests under fitting, indicating that the model initially struggles to capture the training data's nuances. As training continues, the model improves its fit, leading to a lower validation error. However, around two thousand data points, the validation error begins to rise again, signaling overfitting. In summary, the learning curve indicates overfitting, where the model's excessive focus on the training data impairs its ability to generalize. The widening gap between training and validation errors reflects this issue. Strategies like regularization and early stopping can help mitigate overfitting. With a substantial dataset of fifteen thousand measurements, AdaBoost can leverage this rich source of information. The abundance of data allows the algorithm to learn robust patterns and make accurate predictions. It's less likely to fall into the trap of overfitting, as there is enough data to discern genuine patterns from noise. AdaBoost's adaptability is high if the dataset contains complex relationships and interactions between air quality parameters. The algorithm can capture intricate patterns effectively, making it well-suited for challenging datasets. AdaBoost is known for handling imbalanced datasets well by assigning higher weights to misclassified samples, which helps in learning minority classes. On the other hand, irrelevant or noisy features can reduce performance and

potentially lead to overfitting [2, 3, 10, 11, 12]. AdaBoost is generally less prone to overfitting compared to some other complex models, trying to fit the anomalies if the dataset is noisy or contains outliers. This can lead to overfitting, where the model learns the training data too closely and struggles to generalize. To reduce overfitting in AdaBoost, regularization techniques can be applied. Regularization adds a penalty to the loss function, discouraging the model from fitting noise. Common techniques include adjusting the learning rate or limiting the depth or complexity of the base learners. Utilizing cross-validation is essential to ensure AdaBoost's generalization capabilities. It helps identify when overfitting occurs and allows the fine-tuning of the regularization parameters [2, 3, 10, 11, 12]. AdaBoost is known for its efficiency during the training phase. It builds a strong ensemble model by iteratively training weak learners, typically decision trees, on weighted subsets of the data. Since these learners are relatively simple, the training process is often fast and can scale well even to large datasets. Once trained, AdaBoost's predictions are generally quite fast. The efficiency is given by the fact that the model consists of a weighted combination of weak learners, and the evaluation of these learners for prediction is typically computationally inexpensive. AdaBoost's models tend to be lightweight compared to some other complex models like deep neural networks. The ensemble consists of a collection of simple base learners with associated weights. This means that storing and deploying the model is efficient in terms of memory usage and inference speed. Another advantage of AdaBoost is that the training of weak learners in each iteration can be parallelized. This means that, depending on the computational resources available, AdaBoost can take advantage of parallel processing to speed up training even further. AdaBoost's computational efficiency makes it suitable for a wide range of applications, from small to large datasets. It can handle datasets with thousands or even millions of data points, making it versatile for various ML techniques [2, 3, 10, 11, 12]. In table 6 is presented the pseudocode for AdaBoost ML algorithm, while the algorithm implementation in Python is presented in Figure 9 and Figure 10.

Table 6. *AdaBoost algorithm (pseudocode) [1]*

(1) AdaBoost algorithm:
(2) start
(3) initialize training data D and the number of iterations T;
(4) initialize weights for data points: w[i] = 1/N, where N is the number of data points
(5) for t <- 1 to T do;
(6) train a weak learner, such as a decision stump, on the weighted data D with weights w;
(7) calculate the weighted error rate epsilon_t of the weak learner on D;
*(8) calculate the importance of the weak learner: alpha_t = 0.5 * ln((1 - epsilon_t) /epsilon_t);*
(9) update the weights for data points;
(10) for i <- 1 to N do
(11) if the weak learner misclassifies data point i then
*(12) w[i] <- w[i] * e^(alpha_t);*
(13) else
*(14) w[i] <- w[i] * e^(-alpha_t);*
(15) end if;
(16) end for;

(17) normalize the weights so that they sum to 1: w <- w / sum(w);
(18) end for;
(19) combine the weak learners into a strong classifier using their weighted votes;
(20) end.

The research of the performance of machine learning algorithms indicated that AdaBoost and Gradient Boosting are the most suitable algorithms for air quality prediction, closely followed by SVR and Random Forest. While K-Nearest Neighbors performed rather well, it was the least successful of the algorithms tested. These findings are useful in constructing a new air quality prediction tool that uses ML to give real-time air quality predictions.

CONCLUSIONS

In an era of growing environmental concerns, the necessity for accurate, real-time air quality data has become critical. Poor air quality has a direct and considerable influence on human health and well-being, making reliable data available to individuals and authorities for informed decision-making processes [14, 21, 24]. The paper demonstrates the effectiveness of ML algorithms in forecasting and monitoring air quality. Valuable insights were gathered by a thorough examination of five unique algorithms, respectively K-NN, Random Forest, Gradient Boosting, SVR and AdaBoost combined with rigorous cross-validation. The results show that, while all the algorithms performed well, AdaBoost is the best algorithm for air quality prediction, closely followed by Gradient Boosting. The SVR and Random Forest methods performed well, however K-NN had much lower predictive power for air quality prediction. These findings highlight the significance of algorithm selection in making accurate air quality predictions. Furthermore, the application's systematic workflow which includes user interaction, data retrieval from a large database, ML model execution, results visualization, and transparent communication of findings via the user interface, increases its value as a tool for both individuals and authorities. The author’s main contributions are:

- The application code, architecture design development and fine- tuning;
- The implementation of five ML algorithms using the Scikit-learn library from Python and the integration of ML models into the application's framework;
- The identifying of the most suitable ML algorithm for air quality prediction;
- Extensive data analyses and the handling of aberrant data.

The convergence of ML and environmental monitoring offers not just improved air quality forecasts, but also a better knowledge of the variables driving air quality. The developed application exemplifies the ability of the technology in solving major environmental issues, allowing individuals and communities to make the best decisions for a cleaner and safer future.

REFERENCES

[1] Arthur C., Air Pollution, Measuring, Monitoring and Surveillance of Air Pollution, Volume 3, pp.44-723, 1977.

[2] Bishop, C. M. Pattern Recognition and Machine Learning, Springer, New York, pp. 75-602, 2006.

[3] Duda, R. O., Hart, P. E., Stork, D. G., Pattern classification, Second edition, Wiley-Interscience, pp. 102-622, 2000.

[4] Friedman, J. H., Hastie, T., Tibsherany, R., The Elements of Statistical Learning: Data Mining, Inference and Prediction, Springer, pp.22-541, 2001.

[5] Géron, A., Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, O'Reilly Media, pp.42-490, 2019.

[6] Glynn, K., Wade, C., Hands-On Gradient Boosting with XGBoost and Scikit-learn: Perform Accessible Machine Learning and Extreme Gradient Boosting with Python. Packt Publishing, Limited, pp.60-274, 2020.

[7] Jones, B., Beazley, D., Python Cookbook, O'Reilly Media, Incorporated, pp.376-652, 2013.

[8] Lutz, M., Learning Python, O'Reilly Media; 5th edition, pp. 723-1415, 2013.

[9] McKinney, W., Python for Data Analysis: Data Wrangling with Pandas, NumPy, and Python. O'Reilly Media; 2nd edition, pp.103-446, 2017.

[10] Muller, A. C., Guido, S., Introduction to Machine Learning with Python, O'Reilly Media, Inc, USA, pp.27-347, 2016.

[11] Murphy, K, P., Machine Learning: A Probabilistic Perspective, The MIT Press; Illustrated edition, pp.114-975, 2012.

[12] Zelle, J. M., Python Programming: An Introduction to Computer Science, Franklin, Beedle & Associates, pp.100-407, 2003.

[13] Airly, https://airly.org/map/en/.

[14] AirNow., https://www.airnow.gov/.

[15] ANPM, http://www.anpm.ro/.

[16] IQAir, https://www.iqair.com/us/.

[17] Kaggle,https://www.kaggle.com/datasets/ajetkharbri/romania-aqi-2022-1-to-2023-5

[18] Ministerul Mediului. http://www.mmediu.ro/.

[19] National Academies of Sciences, Engineering, and Medicine, https://www.nationalacademiesorg/topics/environment-and-environmental-studies.

[20] United Nations, https://www.un.org/.

[21] UN Environment Programme. https://www.unep.org/.

[22] US EPA, https://www.epa.gov/.

[23] WAQI, https://waqi.info/.

[24] World Health Organization, https://www.who.int/news-room/fact-sheets/detail/ambient-(outdoor)-air-quality-and-health.

Received: December 2023; Accepted: January 2024; Published: January 2024