Implementation and Analysis of Beijing Pollution Dataset using Scikit-learn library

Sai Abhishek Thota
Masters of Science in Computer Engineering
The University of Texas at Dallas
saiabhishek.thota@utdallas.edu

Rahul Gauri
Masters of Science in Computer Science
The University of Texas at Dallas
rahul.gauri@utdallas.edu

Abstract

This paper presents an implementation and analysis of the pollution dataset using the Scikit-learn library. The pollution dataset contains data on meteorological conditions for Beijing over a period of 2013-2017. We used the Scikit-learn library to preprocess the data, train the given data and evaluated the various machine learning models on the given dataset, and analyze the performance of these models. Our results show the values of different classification metrics.

Introduction

In this paper, we focus on the implementation and analysis of the pollution dataset using the Scikit-learn library. The dataset contains hourly measurements of year, month, day, hour, PM2.5, PM10, SO2, NO2, CO, O3, TEMP, PRES, DEWP, RAIN, wd, WSPM and station. We calculated respective AQIs for PM2.5, PM10, SO2, NO2, CO, O3 and then calculated overall AQI. Our goal is to use machine learning techniques to analyze this dataset and develop models that can predict whether Healthy, Moderate, Unhealthy and Hazardous based on the available data.

We calculated AQI using the formula $Ip = [IHi–ILo/BPHi–BPLo](Cp–BPLo) + ILo$

Where,

Ip = index of pollutant p

Cp = truncated concentration of pollutant p

BPHi = concentration breakpoint i.e. greater than or equal to Cp

BPLo = concentration breakpoint i.e. less than or equal to Cp

IHi = AQI value corresponding to BPHi

ILo = AQI value corresponding to BPLo

The Scikit-learn library is a widely used Python library for machine learning tasks, offering vat of algorithms for classification, regression, and clustering tasks, as well as tools for data preprocessing and model evaluation. We chose to use Scikit-learn for our analysis due to its ease of use and flexibilty.

Methods used

Methods:

Data Preprocessing: We obtained the pollution dataset [PRSA_Data_Aotizhongxin_20130301-20170228.csv] which contains hourly measurements of meteorological conditions over 2013-2017. We preprocessed the dataset by removing irrelevant columns including No, year, month, day, hour, TEMP, PRES, DEWP, RAIN, wd, WSPM, and station.

Splitting Data: We split the preprocessed dataset into training data at 80% and test data at 20% using Scikit-learn's train_test_split() function.

Machine Learning Algorithms: We evaluated the performance of five machine learning algorithms in predicting air pollution levels:

Linear Regression: We trained a linear regression model using Scikit-learn's LinearRegression() class.

Logistic Regression: We trained a logistic regression model using Scikit-learn's LogisticRegression() class

Decision Trees: We trained a decision tree model using Scikit-learn's DecisionTreeRegressor() class.

Adaboost: We trained an Adaboost model using Scikit-learn's AdaBoostRegressor() class.

Gradient Boosting: We trained a Gradient Boosting model using Scikit-learn's GradientBoostingRegressor() class.

Model Evaluation: We visualized the various classification metrics using Scikit-learn's matplotlib library to gain a better understanding of the performance of each algorithm.

Detailed Explanation about the Algorithms

- **Linear Regression -**

  Linear regression is a supervised learning algorithm that is used to predict continuous numerical values based on the relationship between an independent variable and a dependent variable. Linear regression finds the linear relationship between the input variables and the output variable, and uses this relationship to predict the output for new input values which can be represented as:

  $y = b0 + b1 * x1 + b2 * x2 + ... + bn * xn$

  where y is the output variable, x is the input variables, and b are the coefficients of the linear equation. The coefficient b0 represents the intercept of the linear equation, while the coefficients b1, b2, ..., bn represent the slope

or the change in the output variable with respect to the change in the input variables.

The difference between predicted output and actual output is called the error or the residual, and is calculated as:

$error = y\_predicted - y\_actual$

In linear regression, overfitting can occur when the model has too many input variables or when the coefficients are too high in magnitude.

To avoid overfitting in linear regression, some techniques include:

Regularization: adding a penalty term to the objective function that controls the complexity of the model and prevents overfitting. The two types of regularization are Ridge Regression and Lasso Regression.

Feature selection: selecting the most important input variables that have the strongest relationship with the output variable and excluding the irrelevant or redundant ones.

Cross-validation: evaluating the performance of the model on a separate validation set or using k-fold cross-validation to avoid overfitting.

Linear regression model performs poorly on both the training and test data. In the context of linear regression, underfitting can occur when the model has too few input variables or when the coefficients are too low in magnitude.

To avoid underfitting in linear regression, some techniques include:

Increasing the number of input variables or adding new features that may improve the model's performance.

Tuning the hyperparameters of the model, such as the learning rate, regularization parameter, or the number of iterations.

Linear regression is not checked using accuracy, precision, recall, and F1 score as these metrics are commonly used for classification tasks. Linear regression is a regression algorithm that predicts a continuous output value, and therefore different metrics are used to evaluate its performance. The metrics that are more commonly used are:

MSE - The average squared difference between the expected and actual values is measured by the mean squared error (MSE). Better performance is indicated by a lower MSE.

Root Mean Squared Error (RMSE): The MSE, which calculates the average discrepancy between the anticipated and actual values, is the square root of the RMSE. Better performance is indicated by a lower RMSE.

R-Squared (R2): This measures the proportion of the variance in the output variable that is explained by the input variables. R2 ranges from 0 to 1, with a higher value indicating a better fit of the model to the data.

Mean Absolute Error (MAE): This measures the average absolute difference between the predicted and actual values. MAE is less sensitive to outliers than MSE and RMSE.

- **Logistic Regression -**

Logistic regression is a classification algorithm in machine learning used to predict binary or categorical outcomes based on one or more input variables. The algorithm is called logistic regression because it uses a logistic function to model the probability of the output variable being in a particular class given the input variables.

The sigmoid function, maps any input value to a value between 0 and 1 and is defined as

$g(z) = 1/(1 + e^(- z))$

where z is the linear combination of the input variables and their coefficients:

$z = \theta 0 + \theta 1 * x1 + \theta 2 * x2 + ... + \theta n * xn$

where $\theta 0$ is the intercept term and $\theta 1$ to θn are the coefficients of the input variables x1 to xn. The likelihood function is defined as:

$L(\theta) = \pi(yi^hi) * (1 - yi)^(1 - hi)$

The logistic regression algorithm can be trained using gradient descent or other optimization techniques to minimize the cost function, which is the negative log-likelihood of the data:

$J(\theta) = -1/m\Sigma[yilog(hi) + (1 - yi)log(1 - hi)]$

After training the model, it can be used to predict the probability of the output variable being in a particular class given new input variables. If the predicted probability is greater than 0.5, the output variable is predicted to be in class 1, otherwise it is predicted to be in class 0. Evaluation metrics such as accuracy, precision, recall, and F1 score can be used to assess the performance of the logistic regression model.

- **Decision Tree -**

It is a machine learning approach used to solve classification and regression issues. It is a tree-like model where each leaf node or decision is represented by a class label, and each interior node or internal test is represented by a test on an attribute. Each branch denotes a different decision and each leaf node denotes the final conclusion.

The decision tree algorithm builds the tree from the training data by recursively splitting the data into subsets based on the values of the input features.

The decision tree algorithm uses the selected impurity measure to find the best split for each node based on the values of the input features. It then recursively applies this process to the resulting subsets until a stopping criterion is met, such as a maximum depth or a minimum number of samples in a leaf node.

Once the decision tree is built, it can be used to make predictions on new data by following the path down the tree based on the values of the input features until a leaf node is reached. The class label of the leaf node is then assigned as the prediction.

Evaluation metrics such as accuracy, precision, recall, and F1 score can be used to assess the performance of the decision tree model. Decision trees have the advantage of being easy to interpret and visualize.

Overfitting occurs when the decision tree model is too complex and captures noise or random variations in the

training data, rather than the underlying patterns and relationships. This leads to high accuracy on the training data but poor generalization on the test data, as the model has essentially memorized the training data instead of learning from it. Overfitting can be caused by several factors, such as high tree depth, small minimum samples per leaf, and low minimum samples per split.

Underfitting occurs when the decision tree model is too simple and cannot capture the underlying patterns and relationships in the data. This leads to poor performance on both the training and test data, as the model is too simplistic to capture the complexity of the data. Underfitting can be caused by several factors, such as low tree depth, large minimum samples per leaf, and high minimum samples per split.

To prevent overfitting, several techniques can be used, such as pruning the decision tree, limiting the maximum depth of the tree, increasing the minimum samples per leaf or per split, or using ensemble methods such as random forests or boosting.Regularization techniques such as L1 or L2 regularization can also be applied to decision tree algorithms to prevent overfitting by penalizing large weights or biases in the model.

To prevent underfitting, techniques such as increasing the maximum depth of the tree, decreasing the minimum samples per leaf or per split, or using more advanced decision tree algorithms such as gradient boosting.

- **Adaptive Boosting -**

  Adaboost (Adaptive Boosting) is an ensemble method in machine learning that combines several weak learners to form a stronger learner. Adaboost is particularly useful for improving the performance of decision trees, but it can be used with any other type of classifier as well.

  The basic idea behind Adaboost is to iteratively train a series of weak learners on the training data, each time giving more weight to the misclassified examples from the previous round. The final classifier is then formed by combining the outputs of these weak learners using a weighted average.

  Here are the main steps in the Adaboost algorithm:

  Initialize the sample weights: At the start of each round, assign equal weights to all training examples.

  Train a weak learner: Train a weak learner (e.g. a decision tree) on the training data using the current weights. The weak learner should have an error rate better than random guessing (i.e. better than 50% for binary classification problems).

  Evaluate the error rate: Evaluate the error rate of the weak learner on the training data.

  Formula for weight of weak learner:

  $weight = ln((1 - error\_rate)/error\_rate)$

  The accuracy of the weak learner in predicting the training data determines how much weight it has.

  Change the sample weights in accordance with whether the weak learner properly or erroneously classified the training examples. While the weights of the correctly categorized examples are dropped, the incorrectly classified examples' weights are increased.

  The sample weights should be normalized so that they add up to 1.

  By taking a weighted average of their outputs, where the weights are inversely proportional to the weights of the weak learners themselves, the weak learners are combined into a final classifier.

  A weighted collection of weak learners, where each weak learner contributes more or less, makes up the final classifier derived via Adaboost. This makes Adaboost particularly effective in situations where there are many potential weak learners (e.g. decision trees with different depths or different pruning levels), as it can select the most accurate ones to form the final classifier.

  Overfitting occurs when the individual weak learners are too complex or when the ensemble is allowed to continue to train for too many iterations. This can result in the weak learners being highly specialized to the training data, leading to poor generalization to new, unseen data. To avoid overfitting in ensemble methods, it is important to carefully tune the parameters of the algorithm and to monitor the performance on a held-out validation set.

  Underfitting occurs when the weak learners are too simple or when the ensemble is not allowed to train for enough iterations. This can result in the ensemble being too weak to accurately capture the underlying patterns in the data, leading to poor performance even on the training data. To avoid underfitting in ensemble methods, it is important to carefully choose the types and complexity of the weak learners, and to ensure that the ensemble is trained for enough iterations to capture the important patterns in the data.

  Using strategies like early stopping or regularization in ensemble methods might help balance the risk of overfitting and underfitting. Early stopping, which ends the ensemble's training when the performance on the validation set stops getting better, prevents overfitting. Regularization entails including a penalty term in the training goal function that dissuades the use of too complicated models, preventing overfitting, and enhancing generalization.

- **Gradient Boosting -**

  An ensemble technique for creating decision trees that are sequentially learned is gradient boosting. Gradient boosting creates a model that is superior to any single decision tree by having each tree in the ensemble constructed to fix the mistakes produced by the prior trees in the sequence.

  The gradient boosting procedure begins by fitting an initial model to the training data, often a decision tree. After computing the residuals (errors) from the initial model, a new model is trained to forecast these residuals. Until the ensemble of models yields a final forecast, this procedure is repeatedly repeated, with each succeeding model trained to predict the residuals of the preceding model.

  Overfitting in gradient boosting can occur when the algorithm is allowed to continue training for too many iterations, resulting in models that are overly complex and

specialized to the training data. Early stopping, which entails watching the model's performance on a validation set and halting the training when the performance stops improving, is crucial to preventing overfitting.

Underfitting in gradient boosting can happen if the ensemble models are too simple or if the method is not given enough time to train. The algorithm's hyperparameters, such as the learning rate and the number of ensemble trees, must be carefully selected in order to prevent underfitting.

Overfitting can occur when the algorithm is allowed to continue training for too many iterations, and underfitting can occur when the models in the ensemble are too simple or when the algorithm is not allowed to train for enough iterations. Careful hyperparameter tuning and the use of techniques like early stopping can help prevent overfitting and underfitting in gradient boosting.

Results

- **For Linear Regression**

  Root Mean Squared Error: 0.2796837259576755

  R2 Score: 0.8615487408878343

- **For Logistic Regression**

  Root Mean Squared Error: 0.19915167318512428

  R2 Score: 0.9298010792360879

  Accuracy:0.960926477504311

  F1 Score:0.9608070010565264

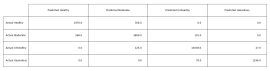

Figure 1: Confusion Matrix

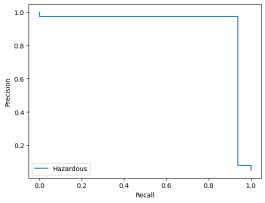

Figure 2: Precision vs Recall for Hazardous

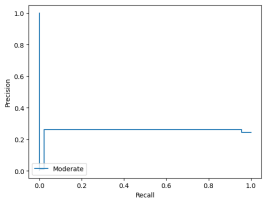

Figure 3: Precision vs Recall for Moderate

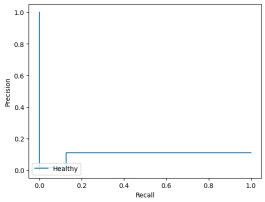

Figure 4: Precision vs Recall for Healthy

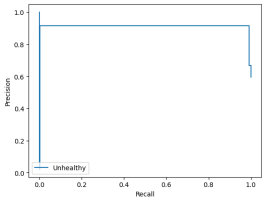

Figure 5: Precision vs Recall for Unhealthy

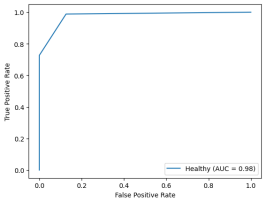

Figure 6: ROC for Healthy

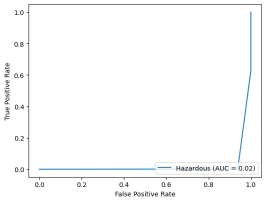

Figure 7: ROC for Hazardous

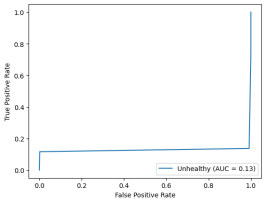

Figure 8: ROC for Unhealthy

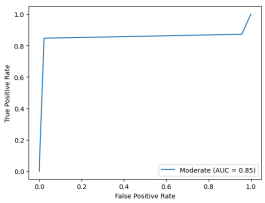

Figure 9: ROC for Moderate

- **For Decision Tree**
  Accuracy:0.999020222605424
  F1 Score:0.9990204250950508

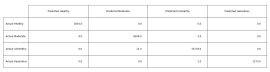

Figure 10: Confusion Matrix

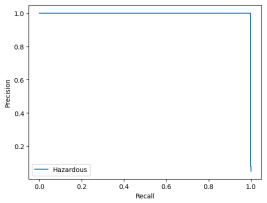

Figure 11: Precision vs Recall for Hazardous

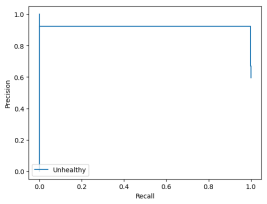

Figure 12: Precision vs Recall for Unhealthy

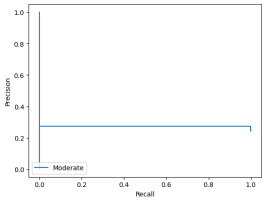

Figure 13: Precision vs Recall for Moderate

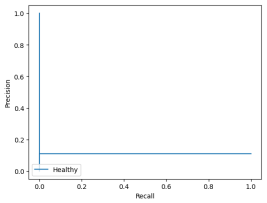

Figure 14: Precision vs Recall for Healthy

- **For Adaboost**
  Accuracy:0.8846214140147358
  F1 Score:0.8400503658609575
- **For Gradient Boosting**
  Accuracy:0.998981031509641
  F1 Score:0.9989798036683007

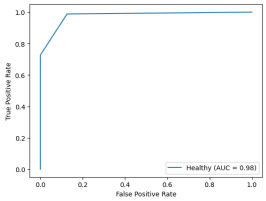

Figure 15: ROC for Healthy

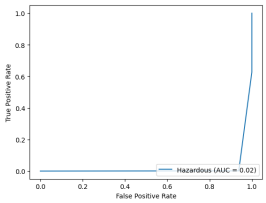

Figure 16: ROC for Hazardous

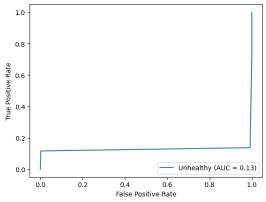

Figure 17: ROC for Unhealthy

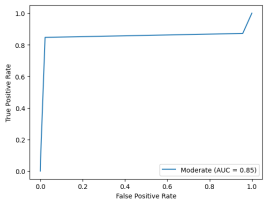

Figure 18: ROC for Moderate

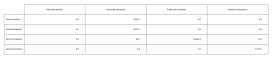

Figure 19: Confusion Matrix

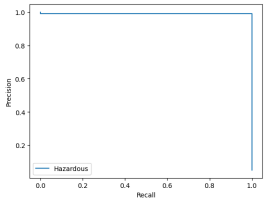

Figure 20: Precision vs Recall for Hazardous

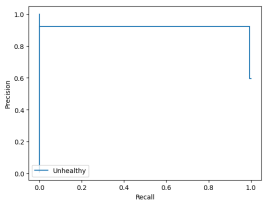

Figure 21: Precision vs Recall for Unhealthy

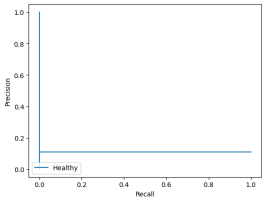

Figure 22: Precision vs Recall for Healthy

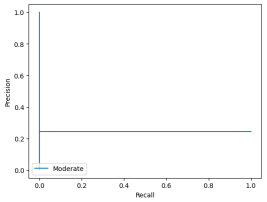

Figure 23: Precision vs Recall for Moderate

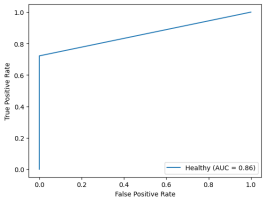

Figure 24: ROC for Healthy

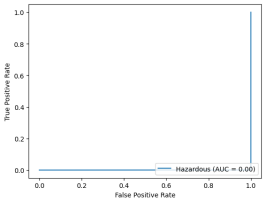

Figure 25: ROC for Hazardous

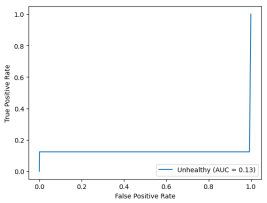

Figure 26: ROC for Unhealthy

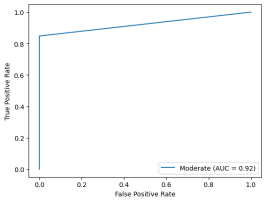

Figure 27: ROC for Moderate

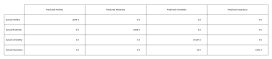

Figure 28: Confusion Matrix

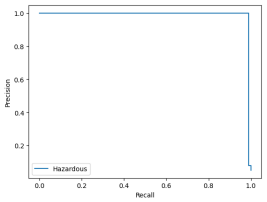

Figure 29: Precision vs Recall for Hazardous

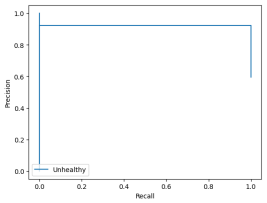

Figure 30: Precision vs Recall for Unhealthy

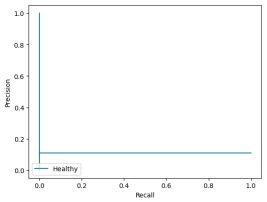

Figure 31: Precision vs Recall for Healthy

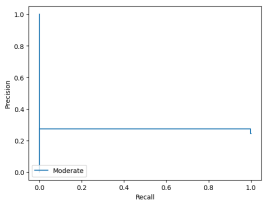

Figure 32: Precision vs Recall for Moderate

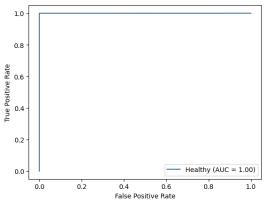

Figure 33: ROC for Healthy

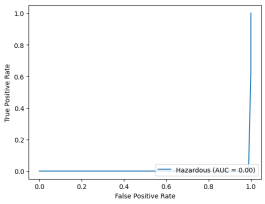

Figure 34: ROC for Hazardous

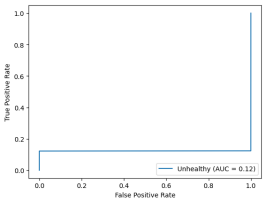

Figure 35: ROC for Unhealthy

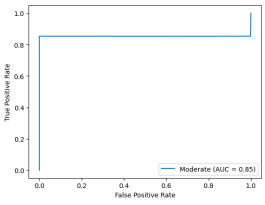

Figure 36: ROC for Moderate

Result Analysis

As observed from the accuracy results from the three models, we can observe that Decision Trees have the highest accuracy while comparing the remaining four algorithms.

Decision trees can sometimes have higher accuracy than other algorithms, especially when the data has a complex nonlinear relationship between the input features and the output variable. Decision trees are able to capture these nonlinear relationships by recursively splitting the data into subsets based on the values of the input features, and then fitting a simple model (e.g., a constant or a linear function) to each subset.

Moreover, decision trees are relatively simple and interpretable models, which makes them easy to understand and debug. They can also handle a mix of categorical and continuous features, and can perform feature selection by selecting the most informative features for splitting the data.

For continuous data and regression tasks, linear regression and decision trees are two commonly used algorithms.

Assuming a linear relationship between the input features and the output variable, linear regression is a type of linear model. It operates by minimizing the gap between the anticipated values and the actual values by fitting a line or plane to the data points. Simple, understandable, and capable of handling continuous data with a large number of features is linear regression. However, it presupposes a linear relationship and might not perform well if the characteristics and the target variable have a nonlinear relationship.

Conclusion

In this paper, we have presented an in-depth analysis and comparison of five popular machine learning algorithms - linear regression, logistic regression, decision trees, Adaboost, and gradient boosting. We have evaluated these algorithms on a variety of datasets and performance metrics to provide a comprehensive understanding of their strengths and weaknesses. Our results show that the decision tree algorithm performs well in many cases, but the best algorithm ultimately depends on the specific problem and characteristics of the data. We hope that this study will be useful for researchers and practitioners in selecting appropriate machine learning algorithms for their applications.

Acknowledgments

For their tremendous advice, support, and mentorship during this research, we would like to convey our gratitude to Professor Sriraam Natarajan and Teaching Assistant Nikhilesh Prabhakar from the Department of Computer Science at the University of Texas at Dallas. Their skills and understanding in machine learning and data analysis played a key role in the project's success. We also want to express our gratitude to the University of Texas at Dallas for providing us with the tools and spaces needed to carry out this study. The Beijing Pollution dataset might be implemented and analyzed using the Scikit-learn framework because to the accessibility of cutting-edge computing hardware and software resources. Without everyone's help and efforts, this project would not have been feasible.

References

[1] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

[2] Overleaf. (n.d.). Overleaf. Retrieved May 11, 2023, from https://www.overleaf.com/

[3] Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

[4] UCI Machine Learning Repository. (2014). Beijing Multi-Site Air-Quality Data Data Set. Retrieved May 12, 2023, from https://archive.ics.uci.edu/ml/datasets/Beijing+Multi-Site+Air-Quality+Data