CS771A Assignment 3

Adit Jain
Junior Undergraduate
Dept. of Electrical Engineering
Roll no.: 200038
`aditj20@iitk.ac.in`

Akshit Verma
Junior Undergraduate
Dept. of Electrical Engineering
Roll no.: 200091
`akshitv20@iitk.ac.in`

Ahmad Nabeel
Junior Undergraduate
Dept. of Material Science Engineering
Roll no.: 200063
`ahmad20@iitk.ac.in`

Abstract

This document is the submission of our group, "No Brainers" for Assignment 3. We have answered Part 1 and 2 with all the relevant level of detail required, providing mathematical proofs wherever required.

1 Solution to Question 1

Below, we have listed down the linear models we tried and the mean absolute error (MAE) we got in each case:

Model	**MAE O_3**	**MAE NO_2**
Linear Regression	6.4896	5.82e^{-15}
Ridge Regression	6.4896	2.56e^{-5}
Lasso Regression	6.4885	0.3583
Elastic Net	6.4918	0.3529
Bayesian Ridge Regression	6.4891	2.47e-14
Passive Aggressive Regression	6.8165	0.0590
Orthogonal Matching Pursuit	7.2985	4.12e^{-15}

As we can clearly see that normal Linear Regression works best for the provided data set if only linear models are allowed, as claimed by the manufacturer. The metric used for evaluation is mean absolute error (the lower the better) on the training data set itself.

One thing we observed was that while implementing linear model for training, we got MAE of O_3 to be much much higher than the MAE of NO_2.

2 Solution to Question 2

While attempting this question, as stated in the assignment, we ignored the timestamp. That is, timestamp was not considered to be a feature while making a prediction. This essentially left us with 6 parameters:

- Humidity
- Temperature
- First output voltage from NO_2 sensor
- Second output voltage from NO_2 sensor
- First output voltage from O_3 sensor
- Second output voltage from O_3 sensor

2.1 Random Forest Regressor

Using these parameters, we initially decided on using a standard Random Forest Regressor. Random Forest Regressor is a type of ensemble model that uses a collection of decision trees to make predictions as explained in the videos provided. In brief, here's how it works in the present scenario:

1. Random Sampling: The Random Forest Regressor randomly selects a subset of the training data (with replacement) to create a "bootstrapped" sample. This sample is used to train an individual decision tree within the ensemble. The remaining data that is not included in the bootstrapped sample is referred to as "out-of-bag" (OOB) data.
2. Decision Tree Training: The model builds multiple decision trees, each using a subset of the features and a different bootstrapped sample from the training data. Each decision tree is trained to predict the target values (outputs) for the corresponding bootstrapped sample.
3. Feature Splitting: For each decision tree, the model selects the best feature and the corresponding split point to create branches that partition the data into subsets based on the values of that feature. This process is repeated recursively, creating a tree-like structure.
4. Tree Ensemble: Once all the decision trees are trained, they are combined to form an ensemble. During prediction, a test data point is passed down each tree, following the splits until it reaches a leaf node, and the average (or weighted average) of the target values in the leaf nodes of all the trees is taken as the final prediction.
5. Model Evaluation: The performance of the Random Forest Regressor is evaluated using evaluation metrics such as MSE, RMSE, and R-squared on the test data. The OOB data, which was not used during training, can also be used for model evaluation to assess its performance.
6. Hyperparameter Tuning: The Random Forest Regressor has several hyperparameters, such as the number of trees in the ensemble, the maximum depth of each tree, and the number of features to consider at each split, among others. Hyperparameter tuning techniques, such as cross-validation, can be used to find the optimal values for these hyperparameters to improve the model's performance. But, we instead used the defaults.

But since the Random Forest Regressor leverages the power of multiple decision trees and their ensemble to make accurate predictions by reducing overfitting and improving generalization, it occupies a lot of disk space and the predictions also take a lot of time. The stats for the Random Forest Regressor model we implemented are as given below:

- Model size: 179.5 MB
- Testing time: 0.2394s for 5,000 rows
- MAE O_3: 1.4100
- MAE NO_2: 0.9419

2.2 KNN Regressor

We now started looking closely at the data inhand. And the best way to do so was to visualize the same. Hence, we plotted the six graphs, one each for the feature, each containing the datapoints from the testing and training datasets. The graphs obtained are as shown in Fig 1.

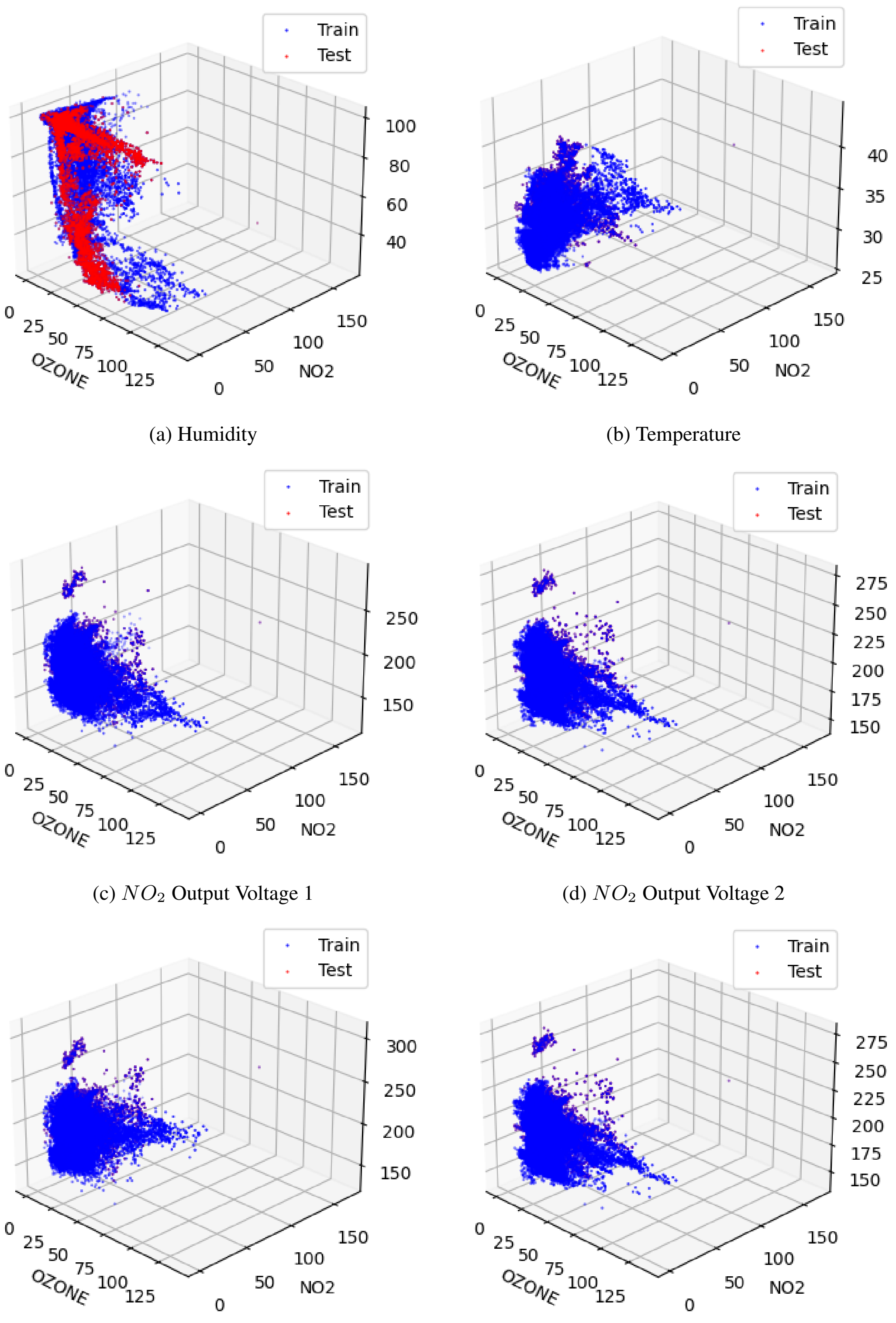
As we can clearly see, the dependence on the parameters is a tad linear for NO_2 but not at all so for O_3. Hence, we need to use more sophisticated and non-linear models. We observed one fascinating thing though, which was the testing and training sets were quite close to each other. This slingshoted us to think about nearest-neighbours since it can be a simple and effective approach for smaller datasets with lower-dimensional features, and it can often capture nonlinear relationships in the data, as is the case with us.

KNN Regressor, or K-Nearest Neighbor Regressor, uses a "lazy learning" approach, where predictions are made based on the values of the k-nearest neighbors in the training dataset. Here's how we used it:

1. Data Preparation: The dataset was divided into a training set and a test set i.e. the `train.csv` and `dummy_test.csv` files provided respectively. The training set is used to train the KNeighborsRegressor model, while the test set is used for evaluating its performance.
2. Feature Scaling: It's important to scale the features in the dataset before using KNeighborsRegressor, as the algorithm calculates distances between data points. Scaling the features ensures that no single feature dominates the distance calculations. Common scaling techniques include min-max scaling or z-score normalization. We used none.
3. Model Initialization: The KNeighborsRegressor model is initialized with hyperparameters, including the value of n_neighbors (=4), and optionally the distance metric to use for calculating distances between data points.
4. Training: During training, the model simply stores the feature values and associated target values (outputs) of the training dataset in memory, without actually building a model or performing any calculations.
5. Prediction: When making predictions for a test data point, the model calculates the distances between the test data point and all the training data points using the chosen distance metric. It then selects the k-nearest neighbors (where k is the value of n_neighbors = 4) based on the smallest distances. The model uses the output values of these k-nearest neighbors, typically by taking the mean or weighted mean of their target values, as the predicted value for the test data point.
6. Repeat for all Test Data Points: The above process is repeated for all the test data points in the testing dataset, and the model makes predictions for each test data point based on the top 4 nearest neighbors.

The stats for the k-Nearest Neighbor Regressor model we implemented are as given below:

- Model size: 1.6 MB
- Testing time: 0.0944s for 5,000 rows
- MAE O_3: 3.0107
- MAE NO_2: 2.0894



(a) Humidity

(b) Temperature

(c) NO_2 Output Voltage 1

(d) NO_2 Output Voltage 2

(e) O_3 Output Voltage 1

(f) O_3 Output Voltage 2

Figure 1: Variation of NO_2 and O_3 with parameters