# **Sprint 3**

In this sprint, we want to train the baseline models with the preprocessed data and use it to predict the CO2 emissions from different types of cars. The target feature to estimate is CO2 (g/km). In the following sections, we present the results of baseline models. We will present the results of two scenarios: with and without using dimensionality reduction techniques.

## **Linear Regression**

As the first model, we trained the Linear Regression model.

### **Linear Regression without Dimensionality Reduction**

The results of applying Linear Regression without reducing the dimension are:

* Mean Squared Error: 2.6485711266259274e-05
* Mean Absolute Error (MAE): 0.0025955619001083283
* Root Mean Squared Error (RMSE): 0.005146427038855139
* R-squared (R2): 0.9944408953741078

A low MSE, MAE, and RMSE indicate that the model's predictions are close to the actual values, while a high R-squared value suggests that the model explains a significant proportion of the variance in the target variable. Overall, these results indicate that the model performs well in predicting CO2 emissions.

### **Linear Regression with Dimensionality Reduction**

We applied PCA to reduce the dimensionality of the dataset. The results of evaluation metrics are:

* Mean Squared Error (MSE): 0.0005988825887608515
* Mean Absolute Error (MAE): 0.017800248244260315
* Root Mean Squared Error (RMSE): 0.024472077736899488
* R-squared (R2): 0.8743001108757107

Reducing the dimensionality with PCA while retaining 95% of the variance has led to faster model application, which is a common advantage of dimensionality reduction techniques. However, the evaluation metrics show that the model's performance slightly decreased after applying PCA compared to the previous model without dimensionality reduction. The mean squared error (MSE), mean absolute error (MAE), and root mean squared error (RMSE) have increased, indicating higher prediction errors. Additionally, the R-squared (R2) value decreased, suggesting that the model explains less variance in the target variable.

## **Ridge Regression**

Ridge Regression is a type of linear regression that incorporates regularization to address high correlation between predictor variables and reduce the model's sensitivity to noisy input data. The results of applying this model are presented in the next sections.

### **Ridge Regression without Dimensionality Reduction**

The results of applying Ridge Regression without reducing the dimension are:

* Mean Squared Error (MSE): 2.9221645265128422e-05
* Mean Absolute Error (MAE): 0.0028743076853023158
* Root Mean Squared Error (RMSE): 0.005405704881431137
* R-squared (R2): 0.9938666482566205

This model seems to perform well based on these metrics, with low errors and a high R-squared value indicating a good fit to the data.

To identify the optimal value of alpha, we used the cross validation. These are the results of the application of the model with optimal value for alpha:

Optimal Alpha: 0.1

* Mean Squared Error (MSE): 2.7492705385780724e-05
* Mean Absolute Error (MAE): 0.0026688016450271846
* Root Mean Squared Error (RMSE): 0.005243348680545737
* R-squared (R2): 0.9942295366678302

It seems that using the optimal alpha value has resulted in improvements in most of the evaluation metrics. The Mean Squared Error (MSE), Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R2) values are slightly better with the optimal alpha, indicating that the model's performance improved after tuning the regularization parameter.

### **Ridge Regression with Dimensionality Reduction**

We used PCA to reduce the dimensionality of the dataset. The results of the application of Ridge Regression model after reducing the dimensions are:

Optimal Alpha: 1

* Mean Squared Error (MSE): 0.0005945705163835583
* Mean Absolute Error (MAE): 0.017751162611685575
* Root Mean Squared Error (RMSE): 0.024383816690246796
* R-squared (R2): 0.8752051747895626

It appears that before applying PCA, the Ridge Regression model with an optimal alpha value of 0.1 achieved slightly better performance compared to after applying PCA. The R-squared value was higher (approximately 0.99), indicating that the model explained more variance in the target variable before PCA. Additionally, the MSE, MAE, and RMSE values were lower, indicating lower prediction errors.

This suggests that the original features without dimensionality reduction might have contained more predictive power for estimating CO2 emissions. However, it's essential to consider the trade-offs between model performance and computational efficiency, as dimensionality reduction techniques like PCA can help reduce the complexity of the model and speed up computation, especially with larger datasets.

## **Lasso Regression**

Lasso Regression, also known as L1 regularization, is a linear regression technique that adds a penalty term to the ordinary least squares objective function. This penalty term is the absolute value of the coefficients multiplied by a regularization parameter (alpha), which controls the strength of regularization.

The main objective of Lasso Regression is to minimize the sum of the squared residuals between the observed and predicted values, similar to ordinary linear regression. However, it also aims to minimize the sum of the absolute values of the coefficients, thereby encouraging sparsity in the coefficient matrix. This means that Lasso Regression tends to produce models with fewer coefficients, effectively performing feature selection by shrinking less important coefficients towards zero.

The regularization parameter, alpha, controls the balance between fitting the data well and keeping the model simple. Higher values of alpha result in more regularization, leading to more coefficients being set to zero and a simpler model.

In summary, Lasso Regression is a useful technique for feature selection and regularization in linear regression models, particularly when dealing with high-dimensional datasets with potentially redundant or irrelevant features.

Using Lasso Regression alongside Ridge Regression is a common practice, especially when dealing with high-dimensional datasets or when you want to perform feature selection. Lasso Regression tends to produce sparse models by setting some coefficients to zero, effectively performing feature selection, whereas Ridge Regression tends to shrink the coefficients towards zero without necessarily setting them exactly to zero.

The results of the application of this model on our dataset are:

* Mean Squared Error (MSE): 0.004764564350626007
* Mean Absolute Error (MAE): 0.05109835557138939
* Root Mean Squared Error (RMSE): 0.06902582379534494
* R-squared (R2): -3.777341135546841e-05

These results show a high mean squared error (MSE), mean absolute error (MAE), and root mean squared error (RMSE), as well as a negative R-squared value. This indicates that the model is performing poorly and may not be capturing the relationship between the features and the target variable effectively.

We tried to tune the value of alpha. The results of this model with the optimal value of alpha are:

Optimal Alpha: 0.01

* Mean Squared Error (MSE): 0.0029877575268617627
* Mean Absolute Error (MAE): 0.0374890930641223
* Root Mean Squared Error (RMSE): 0.05466038352281991
* R-squared (R2): 0.3728974645366506

These results show that the performance of the Lasso Regression model with the optimal alpha value (0.01) improved compared to the previous application. However, the model's performance still seems suboptimal, as indicated by the relatively low R-squared value (0.3729) and the error metrics.

**Therefore, this model is not a good model for our project.**

## **ElasticNet Regression**

ElasticNet Regression is a combination of Ridge and Lasso regression techniques, which combines their penalties. The results of applying this method to the dataset are:

* Mean Squared Error (MSE): 0.004764564350626007
* Mean Absolute Error (MAE): 0.05109835557138939
* Root Mean Squared Error (RMSE): 0.06902582379534494
* R-squared (R2): -3.777341135546841e-05

The results show that the values of MSE, MAE, and RMSE are low. But, the negative value of R-squared indicates that the model performs worse than a horizontal line. This suggests that the model is not able to explain the variance in the data and is performing poorly.

We tried to perform hyperparameter tuning for ElasticNet regression using cross-validation with GridSearchCV. The results are:

* Best Hyperparameters: {'alpha': 0.1, 'l1\_ratio': 0.1}
* Mean Squared Error (MSE): 0.0033337131647687627
* R-squared (R2): 0.30028459159136967

With these hyperparameters, the ElasticNet model achieved an MSE of approximately 0.0033 and an R2 value of approximately 0.3003 on the test set. Overall, while the model's performance improved compared to the previous results, it still may not be satisfactory, especially considering the relatively low R-squared value.

**Therefore, this model is not a good model for our project.**

## **Decision Trees**

Decision Trees are a popular algorithm for regression tasks; therefore, we decided to apply this model on our dataset. We applied this model with and without dimensionality reduction. The results are presented in the next two sections.

### **Decision Trees without Dimensionality Reduction**

The results of this model are:

* Mean Squared Error (MSE): 5.0408947205889875e-06
* R-squared (R2): 0.9989419630502595
* Mean Absolute Error (MAE): 0.00016894822536393775
* Root Mean Squared Error (RMSE): 0.0022451936933344943
* Explained Variance Score: 0.9989419815252438

A very low MSE, high R-squared, and small MAE and RMSE indicate that this model is performing exceptionally well and is able to make accurate predictions with very small errors. Additionally, the explained variance score of nearly 1 suggests that the model is able to explain almost all of the variance in the target variable. Overall, these metrics indicate **that the Decision Tree Regression model is highly effective for the task at hand.**

### **Decision Trees with Dimensionality Reduction**

The results of this model are:

* Mean Squared Error (MSE): 7.293080520500923e-05
* R-squared (R2): 0.9846925018358221
* Mean Absolute Error (MAE): 0.0012739928590183088
* Root Mean Squared Error (RMSE): 0.008539953466208656
* Explained Variance Score: 0.9846934559572341

These results show that the Decision Tree model without dimensionality reduction performed better across all metrics compared to the one with dimensionality reduction.

It should be noted that applying PCA before a decision tree (or ensemble methods like Random Forest) is less common compared to using it with linear models. Decision trees, including Random Forest, are capable of handling high-dimensional data and nonlinear relationships without the need for dimensionality reduction techniques like PCA.

## **Random Forest**

Random Forest is a popular ensemble learning technique that combines multiple decision trees to improve predictive performance and reduce overfitting. It's known for its robustness and ability to handle large datasets with high dimensionality.

The results of applying this model are:

* Mean Squared Error (MSE): 4.391058709938986e-06
* R-squared (R2): 0.9990783575890566
* Mean Absolute Error (MAE): 0.0002066444634440724
* Root Mean Squared Error (RMSE): 0.0020954853160876567
* Explained Variance Score: 0.9990784006100025

These results show that **the Random Forest model seems to perform very well**, achieving low values for MSE, MAE, and RMSE, and a high value for R2, indicating a good fit to the data. The Explained Variance Score is also very high, suggesting that the model explains most of the variance in the target variable.

## **Bagging Algorithms**

Bagging is an ensemble technique where multiple models (often of the same type) are trained on different subsets of the training data. The final prediction is typically made by averaging the predictions of all models (for regression) or using voting (for classification). Random Forest is a popular bagging algorithm based on decision trees. It builds multiple decision trees and merges their predictions to improve accuracy and reduce overfitting. We first applied this model on the dataset. Although the results of this model were satisfactory, we wanted to try some other Bagging models, including Bagged Decision Trees, and Bagged SVM. The results of these models are presented in the following sections.

### **Random Forest**

Random Forest is a popular ensemble learning technique that combines multiple decision trees to improve predictive performance and reduce overfitting. It's known for its robustness and ability to handle large datasets with high dimensionality.

The results of applying this model are:

* Mean Squared Error (MSE): 4.391058709938986e-06
* R-squared (R2): 0.9990783575890566
* Mean Absolute Error (MAE): 0.0002066444634440724
* Root Mean Squared Error (RMSE): 0.0020954853160876567
* Explained Variance Score: 0.9990784006100025

These results show that **the Random Forest model seems to perform very well**, achieving low values for MSE, MAE, and RMSE, and a high value for R2, indicating a good fit to the data. The Explained Variance Score is also very high, suggesting that the model explains most of the variance in the target variable.

### **Bagged Decision Trees**

In Bagged Decision Trees, multiple decision tree models are trained on random subsets of the training data, with replacement. Each tree is built independently, meaning that they can have different splits and structures. During prediction, the output of the bagged ensemble is typically the average (for regression tasks) or the majority vote (for classification tasks) of the predictions made by individual trees. Bagged decision trees are particularly effective when dealing with high variance models, as they reduce overfitting and improve generalization.

The results of applying this model are presented in Table 2. The results show that both Random Forest and Bagged Decision Trees perform quite well across the metrics, with very small values for most error metrics and high values for the R2 and Explained Variance Score. It seems like Random Forest slightly outperforms Bagged Decision Trees in terms of most metrics, but the differences are minimal.

### **Bagged SVM**

Bagged SVM applies the bagging technique to Support Vector Machines. In this approach, multiple SVM models are trained on different subsets of the training data. SVMs are known for their ability to find the optimal decision boundary, but they can be sensitive to the choice of hyperparameters and the specific training data. Bagging helps to reduce the variance of the model by training multiple SVMs on different subsets of the data and averaging their predictions. This can lead to improved performance and robustness, especially in scenarios where the data is noisy or contains outliers.

The results of applying this model are presented in Table 2. The results show that compared to the results of Random Forest and Bagged Decision Trees, Bagged SVM seems to have higher error metrics (MSE, MAE, RMSE) and lower scores for R2 and Explained Variance Score. Therefore, for our project, Random Forest and Bagged Decision Trees might be more effective models than Bagged SVM.

## **Boosting Algorithms**

Boosting is another ensemble technique where multiple weak learners (often shallow decision trees) are trained sequentially, and each subsequent model tries to correct the errors made by the previous one. Gradient Boosting is one of the most popular boosting algorithms. It builds trees sequentially, and each tree tries to correct the errors of the previous one. We applied three boosting algorithms on the dataset, including AdaBoost, Gradient Boosting, and XGBoost.

### **AdaBoost**

AdaBoost is a boosting algorithm that works by iteratively training a series of weak learners (typically decision trees) and combining their predictions to create a strong learner. It adjusts the weights of incorrectly classified instances in each iteration to focus on the difficult examples. The results are presented in Table 2. The results show that AdaBoost perform quite well across most metrics, with low error metrics (MSE, MAE, RMSE) and high scores for R2 and Explained Variance Score. It seems to be a strong performer based on these metrics.

### **Gradient Boosting**

Gradient Boosting is another boosting algorithm that builds a series of decision trees sequentially. Each tree corrects the errors made by the previous ones, with a focus on minimizing the residual errors. The results are presented in Table 2. Comparing the results for AdaBoost and Gradient Boosting show that algorithms perform exceptionally well across these metrics. Gradient Boost, however, appears to have slightly lower error metrics and higher scores for R2 and Explained Variance Score compared to AdaBoost, indicating that Gradient Boost might be slightly more accurate for our project.

### **XGBoost**

XGBoost is a scalable and efficient implementation of Gradient Boosting. It's known for its speed and performance improvements over traditional Gradient Boosting methods. The results of this model are presented in Table 2. These results show that among XGBoost and the two previous algorithms, XGBoost perform the best across all metrics, with the lowest error metrics and the highest scores for R2 and Explained Variance Score. Gradient Boost also performs very well, while AdaBoost performs slightly lower compared to the other two.

## **Deep Learning**

Deep learning models, such as neural networks, can capture complex patterns in data and are suitable for various types of tasks, including regression and classification. We applied different DL techniques that have been presented in the following sections.

### **Multi-layer Perceptron (MLP)**

A Multi-layer Perceptron is a basic type of neural network with multiple layers of neurons (nodes) and non-linear activation functions. The results of applying this technique on the dataset are shown in Table 2. Based on these results, the MLP model appears to perform very well on the task. It exhibits low errors, high explanatory power, and effectively captures the underlying patterns in the data.

### **Convolutional Neural Networks (CNNs)**

CNNs are primarily used for image recognition and processing tasks. They are very effective at capturing spatial patterns in data due to their unique architecture, which includes convolutional layers, pooling layers, and fully connected layers. Although CNNs are suitable for image data, including tasks such as image classification, object detection, and image segmentation, we conducted experiments with different techniques and architectures to see which one yields the best results for our specific use case. The results of applying this technique are shown in Table 2. These results show that this model performs also very well on the task. But, the problem is that training this model for 100 Epoc lasts more than 30 minutes. Therefore, to select between multiple models, we need to consider other factors, including

1. **Model Complexity:** CNNs are typically more complex than MLPs, as they are specifically designed for handling spatial data such as images.
2. **Interpretability:** MLPs often provide more straightforward interpretability compared to CNNs, as they consist of densely connected layers.
3. **Computational Efficiency:** MLPs are generally **faster to train** compared to CNNs, especially for tabular data. If computational resources are limited, an MLP might be a more practical choice.

### **Recurrent Neural Networks (RNNs)**

RNNs are designed to handle sequential data by maintaining a state across time steps. They are commonly used in natural language processing (NLP) tasks, time series analysis, and sequence prediction. The results of applying this technique are shown in Table 2

# **An overview of the results of all models and their interpretation**

This section shows an overview of the results for different models we tried on the dataset. In the first pre-processing step, we applied mean, median, mode, and KNN method to handle missing values. Table 1 shows the results of applying the first six ML model on the cleaned dataset by using these methods. In the next step, we decided to use KNN to handle missing values in all numerical columns. Table 2 shows the results of applying ML models to the dataset cleaned by using this method.

As can be seen in Table 1, the results of ElasticNet Regression are worse than Ridge Regression. As ElasticNet is a combination of Ridge Regression and Lasso Regression, we expect that the results be at least as good as one of the two models. To improve the results of ElasticNet, we use **GridSearchCV** to search for the best combination of **alpha** and **l1\_ratio** hyperparameters for ElasticNet Regression using cross-validation. The best hyperparameters are: {'alpha': 0.01, 'l1\_ratio': 0.1. The results of the ElasticNet model by using these hyperparameters are shown in Table 2.

We also added some further metrics in Table 2, including:

1. **Mean Absolute Percentage Error (MAPE)**: MAPE measures the average absolute percentage difference between the predicted and actual values. It's particularly useful when you want to understand the magnitude of errors relative to the actual values.
2. **Median Absolute Error**: Similar to MAE, but using the median instead of the mean. It's less sensitive to outliers compared to MAE.
3. **Explained Variance Score**: You already have this metric, but it's worth mentioning. Explained Variance Score measures the proportion of variance in the target variable that is explained by the model. It ranges from 0 to 1, where 1 indicates perfect prediction.
4. **Max Error**: Max Error calculates the maximum residual (absolute difference between predicted and true values) across all samples. It gives you an idea of the worst-case error of your model.
5. **Mean Squared Logarithmic Error (MSLE)**: MSLE measures the mean of the squared differences between the natural logarithm of the predicted and true values. It's useful when the target variable has exponential growth.
6. **R-squared Adjusted (R2 Adjusted)**: R2 Adjusted adjusts the R-squared value for the number of predictors in the model. It penalizes model complexity, providing a more reliable measure of goodness-of-fit when comparing models with different numbers of predictors.

Table 1. Results of applying six ML models after the first preprocessing step

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Metric  ML  Model | Mean Squared Error (MSE) | Mean Absolute Error (MAE) | Root Mean Squared Error (RMSE) | R-squared (R2) |
| Linear Regression | 2.6485711266259274e-05 | 0.0025955619001083283 | 0.005146427038855139 | 0.9944408953741078 |
| Linear Regression with PCA | 0.0005988825887608515 | 0.017800248244260315 | 0.024472077736899488 | 0.8743001108757107 |
| Ridge Regression | 2.7492705385780724e-05 | 0.0026688016450271846 | 0.005243348680545737 | 0.9942295366678302 |
| Ridge Regression with PCA | 0.0005945705163835583 | 0.017751162611685575 | 0.024383816690246796 | 0.8752051747895626 |
| Lasso Regression | 0.0029877575268617627 | 0.0374890930641223 | 0.05466038352281991 | 0.3728974645366506 |
| ElasticNet Regression | 0.0033 | 0.05109835557138939 | 0.06902582379534494 | 0.3003 |
| Decision Trees | 5.0408947205889875e-06 | 0.00016894822536393775 | 0.0022451936933344943 | 0.9989419630502595 |
| Decision Trees with PCA | 7.293080520500923e-05 | 0.0012739928590183088 | 0.008539953466208656 | 0.9846925018358221 |
| Random Forest | 4.391058709938986e-06 | 0.0002066444634440724 | 0.0020954853160876567 | 0.9990783575890566 |

Table 2. Results of applying ML models after further optimization of the dataset (rounded to the six number of figures)

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Metric  ML Model | | MSE | MAE | RMSE | R2 | MAPE | MedAE | Max Error | MSLE | R2 Adjusted | Explained Variance Score |
| Linear Regression | | 2.6e-05 | 0.00266 | 0.005086 | 0.99459 | 0.00973 | 0.001631 | 0.091257 | 1.6e-05 | 0.994244 | 0.99459 |
| Linear Regression with PCA | | 0.000592 | 0.017638 | 0.024329 | 0.876186 | 0.062014 | 0.013498 | 0.170917 | 0.000351 | 0.868263 | 0.876187 |
| Ridge Regression | | 2.9e-05 | 0.003036 | 0.005351 | 0.994011 | 0.010805 | 0.002148 | 0.09206 | 1.7e-05 | 0.993628 | 0.994011 |
| Ridge Regression with PCA | | 2.7e-05 | 0.002803 | 0.005201 | 0.994342 | 0.010141 | 0.001923 | 0.092525 | 1.6e-05 | 0.99398 | 0.994342 |
| Lasso Regression | | 0.002982 | 0.037452 | 0.054605 | 0.376282 | 0.140266 | 0.023212 | 0.237502 | 0.001758 | 0.336371 | 0.376283 |
| ElasticNet Regression | | 0.000778 | 0.018575 | 0.027892 | 0.83726 | 0.06638 | 0.011893 | 0.186258 | 0.000448 | 0.826846 | 0.837264 |
| Decision Trees | | 5e-06 | 0.000171 | 0.002197 | 0.99899 | 0.000686 | 0.0 | 0.170642 | 3e-06 | 0.998925 | 0.99899 |
| Decision Trees with PCA | | 7.5e-05 | 0.001316 | 0.008683 | 0.98423 | 0.005516 | 0.0 | 0.255046 | 4.5e-05 | 0.983221 | 0.98423 |
| Bagging Algorithms | Random Forest | 4e-06 | 0.000201 | 0.002062 | 0.99911 | 0.000851 | 0.0 | 0.152991 | 3e-06 | 0.999053 | 0.99911 |
| Bagged Decision Trees | 5e-06 | 0.000219 | 0.00213 | 0.999051 | 0.000938 | 0.0 | 0.150826 | 3e-06 | 0.998991 | 0.999051 |
| Bagged SVM | 0.001666 | 0.031755 | 0.040816 | 0.65152 | 0.115697 | 0.021914 | 0.103569 | 0.000986 | 0.629221 | 0.654758 |
| Boosting Algorithms | AdaBoost | 0.000178 | 0.010845 | 0.013347 | 0.962735 | 0.035307 | 0.008316 | 0.131194 | 0.000102 | 0.960351 | 0.973949 |
| Gradient Boost | 1.2e-05 | 0.001656 | 0.003405 | 0.997574 | 0.006196 | 0.00088 | 0.099021 | 7e-06 | 0.997419 | 0.997575 |
| XGBoost | 4e-06 | 0.000387 | 0.002066 | 0.999107 | 0.001576 | 7.8e-05 | 0.146001 | 3e-06 | 0.99905 | 0.999107 |
| Deep Learning Techniques | MLP | 8e-06 | 0.001266 | 0.002804 | 0.998355 | 0.00474 | 0.000894 | 0.123528 | 5e-06 | 0.99825 | 0.998398 |
| CNN | 1.4e-05 | 0.002796 | 0.003738 | 0.997078 | 0.009604 | 0.002198 | 0.113691 | 8e-06 | 0.996891 | 0.998471 |
| RNN | 0.001337 | 0.023328 | 0.036571 | 0.720235 | 0.081905 | 0.015557 | 0.275887 | 0.000775 | 0.702333 | 0.722463 |