Analyzing Accuracy Improving Techniques by Predicting Nürburgring Lap Times

Jennifer Li, Jenson Ho, Michelle Chow, Andrew Ji, Yuqi Han, Varun Rao

E178 Final Project
Professor Gabriel Gomes
UC Berkeley Spring 2023

1 Abstract

Over the course of this project, our team investigated different techniques to improve accuracy when working on a relatively small data set (100 points). Specifically, the problem we were trying to solve was constructing a predictive model to determine the lap time on a specific race track (Nürburgring Racetrack) based on vehicle characteristics. In order to do this, we set up 3 different models (Linear Regression, Support Vector Regression, and Gradient Boosted Trees). For each of these models, we used 3 different techniques in an attempt to further improve their accuracy (Scaling, Feature Selection, Hyperparameter tuning). While this investigation did not yield any reliably accurate models for lap time prediction (best case average mean squared error yielded 58.55, while our data had a range of 70), it did allow us to observe the affects of the techniques on improving accuracy.

2 Data

2.1 Contents

In order to compile a large data set, we chose the Nürburgring racetrack in Nürburg, Germany for its abundance of recorded track times for a wide array of vehicles. The Nürburgring is about twelve miles with around seventy corners and almost a thousand feet of elevation changes [3]. Additionally, the Nürburgring is one of the few public racetracks that is treated like a regular tolled road, allowing this to be a controlled real-world driving scenario. We utilized a top one hundred list of fastest vehicles recorded on the Nürburgring from 2017 that contained the make, model, year, and lap time [2]. This data set only contains the best lap times for that exact model and specification of the car. There are similar vehicles with different options such as transmission type, which are listed as separate vehicles.

2.2 Variable Selection

Vehicle manufacturer data was added to this data set to obtain parameters to compare with other vehicles. We decided to break down the car into its basic parameters of peak horsepower, peak torque, weight, dimensions, drive type, 0-60 mph, and top speed. Drag was omitted in this study as not all cars had a coefficient of friction available and special trim models came with varying aerodynamic enhancing packages. Tire selection was also omitted due to the complexity of rubber compound, tread, tarmac conditions, and non original equipment tire setups.

Horsepower and torque are two extremely important parameters to a vehicle as they are the main metrics for the total output of a vehicle. Typically horsepower measures how fast the car can do work, while torque measures rotational force and the ability to do work. Depending on the track layout, the track times may favor one parameter over another. Because the Nürburgring has a variety of curves and straights, this gives a more level playing field for all vehicles despite the power gap. Although peak horsepower and torque may indicate a car's ability, the revolutions of the engine when the peak power occurs are important since every vehicle has a varying range of revolutions in which the engine produces more power. This range of power produced by an engine is given in the form of power vs time plotted on a curve [4]. In an ideal case, we would have the power curves of each vehicle tested on the same dynamo-meter; however, this data is kept a secret by the manufacturers unless physically tested. We make up for this lack of the power curve with the 0-60 mph testing time because this enables the model to have a correlation between peak torque and vehicle speed. The top speed similarly aids in comparing horsepower to top speed as peak horsepower occurs closer to the end of the power curve.

The weight of a vehicle significantly impacts its efficiency as more weight requires more force or power to be produced in order to move the vehicle. Most performance vehicles aim to have a good balance between power and weight. Wheelbase plays a large role in how the weight of a car influences its performance. The wheelbase of a car is the measurement between the center of the front tires to the center of the rear tires. In a turn for example, the longer wheelbase of a vehicle would cause the vehicle to have a higher moment of inertia and thus lower cornering speed [9]. Additionally, the height and width of the vehicle help to understand the center of mass as a taller vehicle would typically have a higher center of mass, which can also be attributed to a slow cornering speed as tipping becomes a greater concern.

3 Models

3.1 Linear Regression

Linear Regression is a model used to map a group of variables to one another. In this case, linear regression is used to solve a prediction problem. Given a data set of N input-output pairs, linear regression creates a function that predicts an output $\hat{y}$ while minimizing the error between the actual output and the predicted output. In this case, the inputs are the various variables on each car, mentioned in the Data section, and the outputs are the lap times.

For regressions used in prediction problems, such as this one, the L_2 loss function is used, shown in Equation 2, which returns a positive number that decreases as the model improves. The L_2 loss function is used in place of the L_1 loss function, shown in Equation 1, in linear regression as it is differentiable everywhere, due to being a smooth function, and penalizes large errors due to the squared term. This makes it a more suitable choice for optimization algorithms.

$$L_1 = |y - \hat{y}| \tag{1}$$

$$L_2 : (y - \hat{y})^2 \tag{2}$$

The formula for general regression is shown in equation 3 where D is the number of parameters. In this case, that would be the height, weight, 0-60 time etc. The various θ values are the coefficients that the linear regression model will generate to minimize the L_2 loss function.

$$\hat{y} = h(x; \theta_0, \underline{\theta}_1) = \theta_0 + \theta_1 x^1 + ... + \theta_D x^D \tag{3}$$

It is important to note that in a general case, the regressors themselves do not need to be linear, ie. be of the first power. They are linear in Equation 3 since x is of the first power everywhere but this need not be the case. The model is still considered linear regression as long as the dependent variable $\hat{y}$ is equal to the regressor multiplied by fixed coefficients θ. If the regressor was x^2, for instance, the relationship of the dependent variable $\hat{y}$ would still be linear to the x^2 term. With this in mind, a general form of linear regression can be created, which is shown in Equation 4 where the ϕ terms are functions of x but are still linear to the dependent variable $\hat{y}$.

$$\hat{y} = h(x; \theta_0, \underline{\theta}_1) = \theta_0 + \theta_1 \phi_1(x) + ... + \theta_D \phi_D(x) \tag{4}$$

Linear regression is easily implemented in Python with the sklearn package. The data is first read off the .csv file and cast to a Pandas Dataframe, where it is split into training and testing data. 80% of the data is used for training and 20% is for testing. This ratio can be changed, but too low of a training percentage will not make an accurate model and too low of a testing percentage will make it difficult to see the results of the trained model.

3.2 Gradient Boosted Trees

Gradient Boosted Trees are ensemble machine learning algorithms. Ensemble, or boosting, methods use multiple models that try to use patterns in the data to improve the overall performance of a predictive model. In this case, we combine multiple decision trees and fit a function to the residuals of the previous models.

Decision trees are supervised learning algorithms that use tree-like models to make decisions: it partitions data into sets that correspond to a node in the decision tree. The subsets are created to be as homogeneous as possible with respect to the target variable by partitioning based on values of input features, reducing uncertainty at the node. This is done by choosing an input feature that results in the greatest reduction in entropy or Gini impurity at each node [5]. To assist in measuring the entropy or Gini impurity, we calculate:

$$\rho_k = \frac{\text{\# data points of type k in } \mathcal{D}_\rangle}{\text{\# data points in } \mathcal{D}_\rangle} \tag{5}$$

Entropy is a measure of the uncertainty in the data, and is measured using:

$$H(\mathcal{D}_\rangle) = -\sum_{k=1}^{K} \rho_k \log \rho_k \tag{6}$$

Gini impurity measures the probability of misclassification and is measured using:

$$G(\mathcal{D}_\rangle) = \sum_{k=1}^{K} \rho_k (1 - \rho_k) \tag{7}$$

The weighted impurity of the split can then be calculated using:

$$I(\{\mathcal{D}_1, \mathcal{D}_2\}) = \frac{|\mathcal{D}_1|}{|\mathcal{D}_0|} I(\mathcal{D}_1) + \frac{|\mathcal{D}_2|}{|\mathcal{D}_0|} I(\mathcal{D}_2) \tag{8}$$

By selecting the feature that maximally reduces entropy or Gini impurity, the algorithm can create subsets of data that are as homogeneous as possible with respect to the target variable.

Decision trees have many advantages in that they are multi-class, very fast, and easy to understand. In addition you can control bias-variance through hyperparameters like depth. However, decision trees are not very accurate and tend to overfit, which can be solved through gradient boosting. The decision trees are considered "weak" machine learning models that are combined to compose a "strong" machine learning model [5].

Gradient boosted trees iteratively add decision trees to the model, with each new weak model predicting the "error" of the current strong model, correcting the errors of the previous trees. The process involves optimizing a loss function, shown in Equation 9, which measures the "error" as the difference between the predicted values and the values of the target variable – a regressive label [6].

$$\mathbb{L} = \frac{1}{2} \|\boldsymbol{Y} - \hat{\boldsymbol{Y}}\|_2^2 \tag{9}$$

This is added in by subtracting the weak model from the strong model to reduce the strong model's error:

$$\boldsymbol{Y}_t = -\nabla\mathbb{L}(H_{m-1}(\boldsymbol{X})) \tag{10}$$

$$h_m = train(\boldsymbol{X}, \boldsymbol{Y}_t) \tag{11}$$

$$H_m = H_{m-1} + \sigma h_m \tag{12}$$

where m is each iteration of the algorithm.

Equation 10 calculates the negative gradient of the loss function with respect to the predictions made by the current ensemble of trees on the training data. The negative gradient is used as the target variable for the next tree in the ensemble, such that the next tree tries to correct the errors made by the previous ensemble. $\nabla\mathbb{L}$ is the gradient of the loss function from Equation 9.

Equation 11 trains a new decision tree h_m on the training data $\boldsymbol{X}$ and the negative gradient $\boldsymbol{Y}_t$.

Equation 12 updates the ensemble of trees by adding the newly trained tree h_m with a step size (or learning rate) σ to the current ensemble of trees H_{m-1}. σ is a hyperparameter that controls the contribution of the new tree to the ensemble. By using a learning rate less than 1, the algorithm can prevent overfitting and improve the generalization performance of the model [7].

By minimizing this loss function, we can create a model that predicts the target variable for new data points.

Like linear regression, gradient boosted trees are also implemented in Python with the sklearn package. Data is read off the .csv file and cast to a Pandas dataframe where 80% of the data is split for training and 20% is split for testing. Gradient boosting is then performed with the GradientBoostingRegressor() function.

3.3 Support Vector Machines

Support Vector Machines (SVMs) find the best hyperplane that correctly separates data into two groups. Graphically, this hyperplane can be visualized as the line or plane that physically separates all data points into their two respective groups. Unlike typical feature selection based models, SVMs are based on kernel selection to define the hyperplane. Kernel selection, or kernelization, describes the process of mapping the data into higher dimensions to reveal an easier way to separate the data. This mapping is defined by the kernel function in Equation 13.

$$k(x', x) = \phi(x')^T\phi(x) \tag{13}$$

As proved in class, deriving from Ridge Regression can show that any symmetric and semi-positive definite function guarantees the existence of a feature function, $\phi(x)$, that can be rewritten into the kernel function, $k(x', x)$. This kernel function can then be used to find the kernel matrix $\mathbb{K}$ to solve for α, the distance from each data point to the hyperplane, which ultimately defines the full hyperplane (Equation 15). In summary, SVMs can essentially create their own features to generate a better fitting hyperplane.

$$\mathbb{K} := \Phi\Phi^T \in \mathbb{R}^{N\times N} \text{ where } k(x_i, x_j) = \phi^T(x_i)\phi(x_j) \tag{14}$$

$$\alpha = (\Phi\Phi^T + \lambda I_n)^{-1}Y \tag{15}$$

After defining the shape of the hyperplane, SVM then uses Maximum Margin Classification to define the exact location of the hyperplane. Oftentimes there are several places the hyperplane can lie that all accurately separate the data, however the most optimal location is one where the data points nearest the hyperplane from each group are as far away from the hyperplane as possible. Each data points' distance to the hyperplane is defined as its margin. The larger the margin, the greater the confidence in the classification of the datapoint, hence the name Maximum Margin Classification. To compute this, first assume a dataset that is linearly separable and mapped to one-dimension (x-y plane) so their groups are y={-1, 1}. Compressing the data to one dimension quickly shows that the maximum margin occurs along x at the midpoint between the two groups and when y=0. However, this method can be generalized for any arbitrary dataset: given a line that lies between the two groups, minimize the slope θ to find the points in each group closest to the hyperplane, without crossing into the group. Then, the midpoint of this line will be the location furthest from each group and the optimal location of the hyperplane. This result is represented below.

$$\begin{gathered}\text{minimize } \|\theta\|^2 \\ \text{subject } y_i \alpha(x_i) \geq 1 \text{ for } i = 1 \ldots N\end{gathered}$$

To account for non-separable data, where some data points are on the wrong side of the hyperplane boundary, we include ξ, the distance the data point must travel to return to the correct group. Adding ξ and minimizing this distance, we reach the final form of the Maximum Margin Classifier:

$$\begin{gathered}\text{minimize } \|\theta\|^2 + C \sum_{i=1}^{N} \xi_i \\ \text{subject } y_i \alpha(x_i) \geq 1 - \xi_i \text{ for } i = 1 \ldots N \\ \xi_i \geq 0\end{gathered}$$

All in all, SVMs can improve the accuracy of a model as kernel based selection provides more flexible hyperplanes. Additionally, Maximum Margin Classification only relies on the data points near or on the wrong side of the hyperplane, lessening computational cost and time and of working through the entire data set. These invaluable data points are known as the Support Vectors and are ultimately the only data points that affect the resulting model; giving rise to the name of Support Vector Machines.[8]

4 Methods to Improve Accuracy

4.1 Scaling (Standardization)

Scaling serves to improve model accuracy by modifying the values so that they sit in a similar scope, minimizing the effects that abnormally large or small features may have on the model. In this investigation, scaling was accomplished through standardization, which is given by the following equation.

4.2 Feature Selection (Forward Stepwise Selection)

Feature selection is used to improve model accuracy by removing irrelevant, or negligible features from the data set. This mitigates the affects of overfitting, and removes complexity from the model. Feature selection in this case was accomplished through forward stepwise selection. This involved evaluating the different features individually and determining their accuracy (measured by mean squared error), and then combining them (with priority given to the higher accuracy features) until we are able to attain the best possible accuracy (minimizing mean squared error). This was implemented through the code shown below. The evalFeatures method serves to rank the accuracy of the different features through their mean squared error values, and the efficiencyGraph method serves to conduct forward stepwise selection, returning the best feature set, as well as a graph of how the introduction of the features affected accuracy (in terms of mean squared error).

```
    def evalFeatures(predictor, X_train, X_test, reverse):
        rankedfeatures = {}
        for feature in X_train:
            model = predictor()
            model.fit(X_train[[feature]], Y_train)
            rankedfeatures[mean_squared_error(Y_test, model.predict(X_test[[feature]]
                                                        ))] = feature
        return rankedReturnFeatures
```

```
    def efficiencyGraph(predictor, X_train, X_test, reverse, predictorType):
        features = evalFeatures(predictor, X_train, X_test, reverse)
        keys = list(features.keys())
        MSE = {}
        Predictions = {}
        Params = {}
        for i in range(len(features)):
            EvaluatingCombo = []
            for j in range(i+1):
                EvaluatingCombo.append(features.get(keys[j]))
            model = predictor()
            model.fit(X_train[EvaluatingCombo], Y_train)
            error = mean_squared_error(Y_test, model.predict(X_test[EvaluatingCombo])
                                                        )
        Params[error] = EvaluatingCombo
        Predictions[error] = model.predict(X_test[EvaluatingCombo])
        MSE[error] = model
        plt.plot([None] + list(MSE.keys()))
        plt.xlabel("Number of Features Introduced")
        plt.ylabel("Mean Squared Error")
        plt.title("How Number of Features affects Mean Squared Error for " +
                                                    predictorType)
        plt.show
        BestError = min(list(MSE.keys()))

        return MSE, Params.get(BestError)
```

4.3 Hyperparameter Tuning (Gridsearch Cross Validation)

Hyperparameter tuning improves model accuracy by mechanically varying hyperparameters inorder to determine what combination will work best for the problem at hand. In this investigation, Gridsearch Cross Validation was the method of Hyperparameter tuning. Gridsearch Cross Validation works by taking in a grid (dictionary in python) of hyperparameters with associated values for testing. The system then tests all of the combinations of hyperparameters, evaluating each combination through cross validation, and selecting the best model.

Cross validation during this investigation used the k-fold technique, which involves splitting the training data into k different sections, selecting k-1 of those sections to build a model, and then finally evaluating the model (measured through mean squared error) against the remaining section. This is done with every combination of sections, and the final score generated by this cross validation is an average of these individual scores.

4.3.1 Linear Regression Hyper Parameters

In the case of Linear regression, 2 parameters were varied: regularization, and alpha. Regularization techniques were varied by implementing Gridsearch Cross Validation on a Lasso and then a Ridge linear regression model. Within the parameter grid used to conduct the gridsearch, we decided to vary the alpha and fit intercept values. Alpha serves as a term which regulates regularization strength within the model.

4.3.2 Gradient Boosted Trees

In the case of Gradient Boosted Trees, 2 parameters were varied, alpha and the max leaf nodes value. Alpha serves to regulate the Huber loss function (which works as a combination of mean squared error and absolute error), by designating the threshold between linear (absolute) and quadratic (mean squared) error. The loss function itself serves to build the decision trees. The Max leaf nodes value varies the maximum number of final leaves at the end of the decision tree.

4.3.3 Support Vector Regression

In the case of Support Vector Regression, 2 parameters were varied: C (regularization parameter) and degree (Degree of the kernel function). C is inversely proportional to the regularization strength within the model.

5 Procedure

5.1 Data Processing

We began by collecting data from a data set of the top 100 Nürburgring lap times as of 2017 sourced from Kaggle. Additional parameters were added by scrubbing for the manufacturer vehicle data. All of the data was input onto a Google sheet and then converted into a CSV file. Categorical data was turned into a binary for each option in the category seen in the column.

time (s)	horsepower (hp)	torque (lb-ft)	kerb weight (lb)	wheelbase (in)	width (in)	height	RWD	AWD	FWD	0-60 mph (s)	top speed (mph)	manual	automatic
408	453	280	1664	94	72.83	43	1	0	0	2.8	173	1	0

Table 1: Sample of Data Format Excluding Unused Parameters

5.2 Model Evaluation

For each model, the following was conducted in order to evaluate model accuracy. First, A random seed was chosen in order to keep our results consistent.

```
rng_seed = 425
```

Then, data was split into 80% training data and 20% testing data.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2,
                                    random_state=rng_seed)
```

Then, this stock model is evaluated through the following method, which displays a graph comparing predicted data to the actual data, as well as calculating the mean squared error and coefficient of determination.

```
def evalModel(prediction, modeltype):
    print('Mean squared error (MSE): %.2f' % mean_squared_error(Y_test,
                                          prediction))
    print('Coefficient of determination (R^2): %.2f' % r2_score(Y_test,
                                          prediction))

    plt.scatter(Y_test, prediction)
    plt.xlabel("Testing Lap Time (s)")
    plt.ylabel("Predicted Lap Time (s)")
    plt.title("Predicted vs Testing times using " + modeltype)
    plt.show()
```

Each model was once again evaluated fitting it to the scaled X_train, and Y_train data, generating values and plots similar to the ones generated for the stock model. This fitted data is also used to generate the following modified models.

Feature selection analysis was conducted in a similar manner for each model (using the feature selection algorithm shown in section 4.2). Then the different sets of features (the set with 1 feature, then 2, etc.) were plotted against their respective mean squared errors. Finally, a standard data plot (predicted values vs actual values) as well as the calculated values were generated.

Next, Grid Search Cross validation was conducted on the respective models with different parameters for each model. The specific hyperparameters being varied is detailed in section 4.3. The best models generated through this method are then displayed by the standard plot of predicted vs actual values.

6 Results

6.1 Linear Regression

A summary of results for the performance of Linear Regression is shown in Table 2. Graphs of the test data are shown in Figures 1a-1f of Appendix A.

Technique	Mean Squared Error (MSE)	Coefficient of Determination (R^2)
Base Case	67.97	0.41
Scaling	67.97	0.41
Best Case Feature Selection	58.68	0.49
Lasso Regularization and Hyper Parameter Tuning	61.82	0.47
Ridge Regularization and Hyper Parameter Tuning	58.55	0.49

Table 2: Summary of Performance of Linear Regression Algorithm

6.2 Gradient Boosted Trees

A summary of results for the performance of Gradient Boosted Trees is shown in Table 3. Graphs of the test data are shown in Figures 2a-2e of Appendix B.

Method	Mean Squared Error (MSE)	Coefficient of Determination (R^2)
Base Case	73.05	0.37
Scaling	73.05	0.37
Best Case Feature Selection	73.37	0.37
Grid Search Cross Validation	62.70	0.46

Table 3: Summary of Performance of Gradient Boosted Trees Algorithm

6.3 Support Vector Regression

A summary of results for the performance of Support Vector Regression is shown in Table 4. Graphs of the test data are shown in Figures 3a-3e of Appendix C.

Method	Mean Squared Error (MSE)	Coefficient of Determination (R^2)
Base Case	114.5	0.01
Scaling	95.38	0.18
Best Case Feature Selection	76.70	0.34
Grid Search Cross Validation	63.78	0.45

Table 4: Summary of Performance of Support Vector Regression Algorithm

7 Analysis

Looking at the results of the model evaluations, we observe certain interesting phenomena. Scaling seems to have no effect on the effectiveness of gradient boosted trees and linear regression based models, while having a remarkable affect on support vector regression based models (see tables 2, 3, and 4). This is likely due to how these types of models interact with data, linear regression and support vector regression being unaffected by the scale of the variables involved, whereas gradient boosted trees rely moreso on the scale of variables involved to generate predictions.

Feature selection also yielded some interesting results, having a significant effect on model accuracy for linear regression and support vector regression, while having a marginal effect on gradient boosted trees. Looking at the graphs of effectiveness generated when calculating the best feature sets for the respective models, support vector regression and linear regression benefited from the exclusion of certain variables when building a model. It's also remarkable that support vector regression was able to perform best when exposed to only one of the features (reducing mean squared error by roughly 20 percent). This contrasts with gradient boosted trees based regression, which seemed to benefit from having as much of the data as possible to make decisions.

Hyper parameter tuning proved to be the most reliable technique when it came to improving model accuracy, having a consistent, positive affect on all the models (only having a relatively minor affect on Linear regression). This signals the importance of tuning a model's hyper parameters regardless of the type of model. Often, the stock configuration of the model training system is not the best, and the introduction more specific hyper parameters will drastically improve model accuracy.

Overall, the usage/introduction of these techniques served to improve the accuracy of these models, even with the relatively sparse data given. However, it is also clear that these specific techniques did not have a consistent measure able effects on the different models, ranging from a profound affect (nearly 50 percent reduction in mean squared error for support vector regression) to relatively small effects (roughly 10 percent reduction in mean squared error for linear regression).

8 Conclusion

In conclusion, we used three different models — Linear Regression, Gradient Boosted Trees, and Support Vector Regression — to model lap times on the Nurburgring track, and analyze the effects of data preparation techniques on models built on small data sets. Even though the investigation itself could not give a reliable and accurate model for the lap time prediction due to small data size of 100 points, the comparison between those three models and applied accuracy improving methods did shed lights on optimizing the method when making predictions based on small scale of data by showing interactions between model chosen and accuracy improving methods. We could select the most suitable method of accuracy improvement techniques according to the data analysis model and the variable scale. In the end, the model that had the best performance was the tuned ridge regression model using the best set of features for linear regression.

This investigation revealed the importance of preparing data in order to best complement the model, as well as working to tune the model to the data in order to improve performance. The affects of these accuracy improving techniques especially pronounced in this application, showing how they help combat over fitting, while decreasing the complexity of the model. Though this investigation has provided insight into the affects of these techniques, it was conducted within a hyper specific environment (small data set, relatively few techniques investigated), and our understanding would greatly benefit from further study.

Specifically, further avenues of investigation are present for each of the different accuracy improving techniques. In this investigation, we used data standardization as our means of scaling, and only applied it to the input variables. Scaling could be further investigated by looking at the effects of output scaling, and analyzing how normalizing the data may contribute to accuracy. When conducting feature selection, we used forward step wise selection, which would generate different feature sets based off of precedence (first set : first best feature, second set : first two best features, etc.). This does not provide insight into how the other excluded possible sets (that aren't based on precedence) might perform. Finally, it would be enlightening to explore the effects of varying hyper parameters outside of those already explored.

References

[1] Gomes, G. ”E178/E278/ME292b Statistics and Data Science for Engineers Reader” UC Berkeley, 2023.

[2] ”Top 100 Track Times.” 2017, www.nurburgringlaptimes.com/lap-times-top-100/ .

[3] ”Nurburgring Nordschleife - Track Review.” MotorTrend, 2011, www.motortrend.com/features/modp-1104-nurburgring-nordschleife-track-review/.

[4] Hilgendorf, D. ”Difference Between Horsepower and Torque.” AMSOIL, 2023, www.blog.amsoil.com/the-difference-between-horsepower-and-torque/.

[5] Gomes, G. ”Lecture 14 SDSE Decision Trees.pdf” UC Berkeley, 2023.

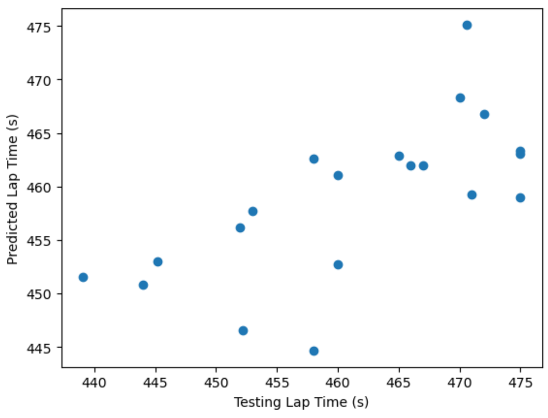
[6] Gomes, G. ”Lecture 16 SDSE Gradient Boosting.pdf” UC Berkeley, 2023.

[7] Gaurav. “An Introduction to Gradient Boosting Decision Trees.” Machine Learning Plus, March 8, 2022. https://www.machinelearningplus.com/machine-learning/an-introduction-to-gradient-boosting-decision-trees/.

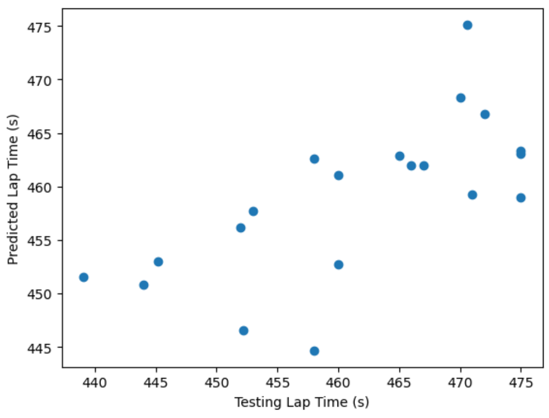
[8] Gomes, G. ”Lecture 13 SDSE SVM video.pdf” UC Berkeley, 2023.

[9] Weihua Zhang, ”Chapter 5 - Optimization design method for the dynamic performance of high-speed trains.” Elsevier, 2020, ISBN 9780128133750, https://doi.org/10.1016/B978-0-12-813375-0.00005-4.

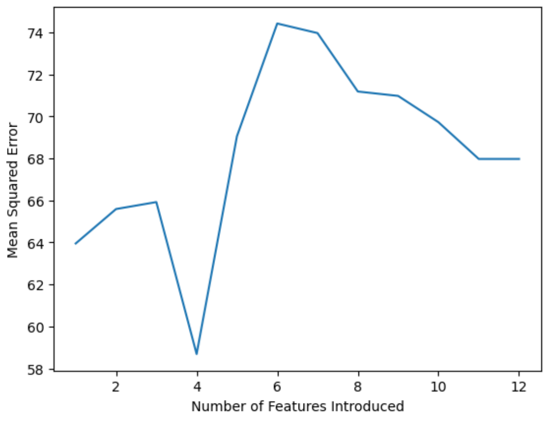
Appendix A Linear Regression Graphs



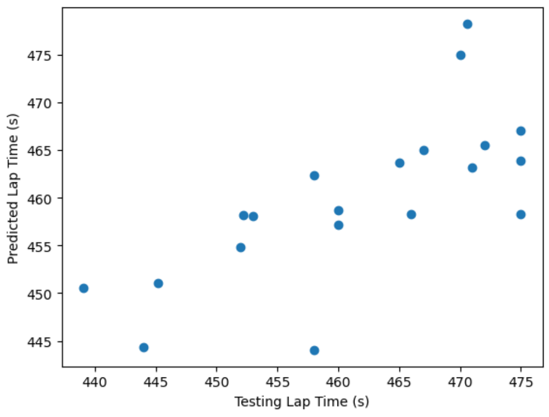
(a) Predicted vs Testing Times Using Linear Regression (Base Case)



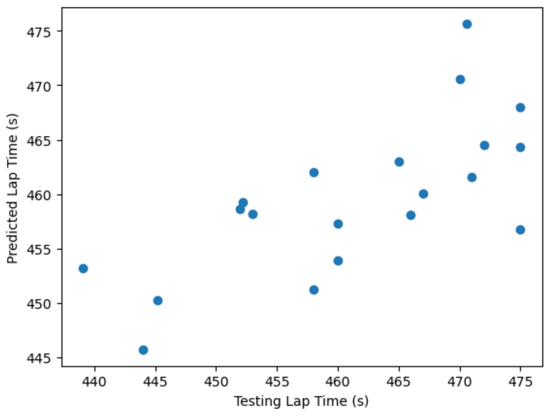
(b) Predicted vs Testing Times Using Linear Regression (Post Scaling)



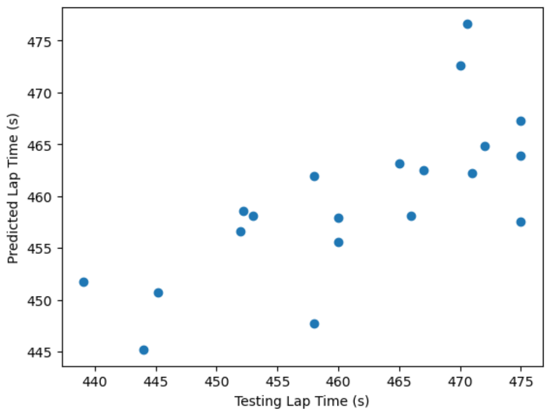
(c) How Number of Features Affects Mean Squared Error for Linear Regression



(d) Predicted vs Testing Times Using Linear Regression Using Best Case Feature Selection



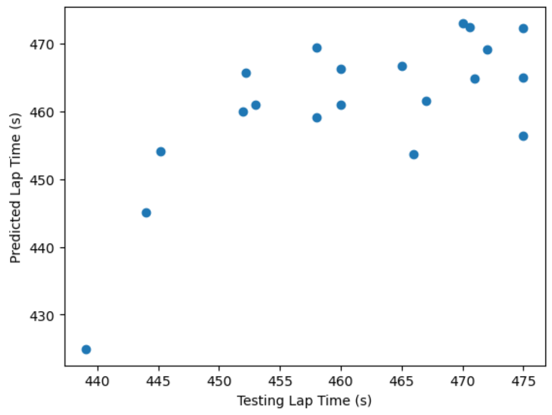
(e) Predicted vs Testing Times Using Linear Regression With Lasso Regularization and Hyperparameter Tuning



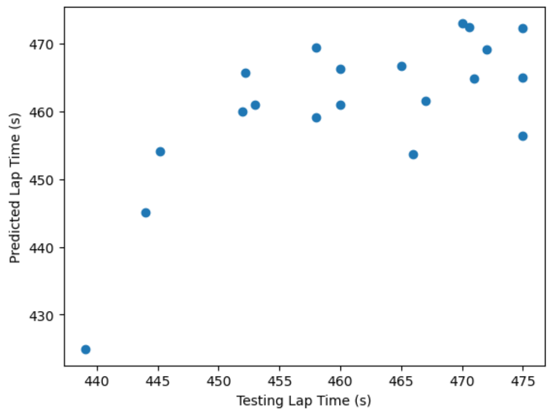
(f) Predicted vs Testing Times Using Linear Regression With Ridge Regularization and Hyperparameter Turning

Figure 1: Summary of Graphs for Linear Regression Algorithm

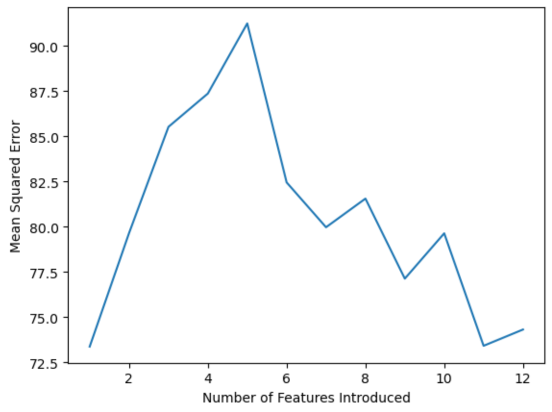
Appendix B Gradient Boosted Trees Graphs



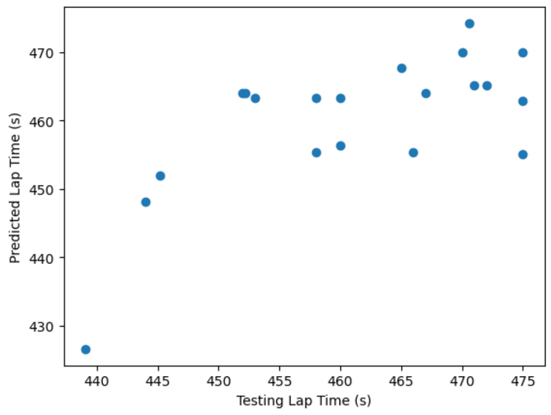
(a) Predicted vs Testing Times Using Gradient Boosted Trees (Base Case)



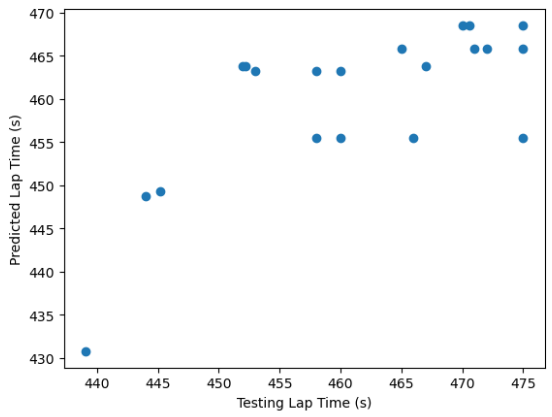
(b) Predicted vs Testing Times Using Gradient Boosted Trees (Post Scaling)



(c) How Number of Features Affects Mean Squared Error for Gradient Boosted Trees



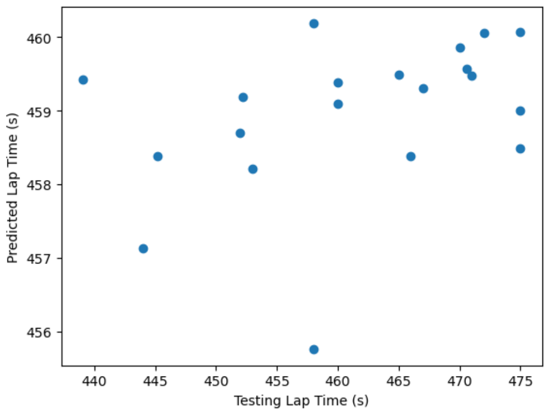
(d) Predicted vs Testing Times Using Gradient Boosted Trees Using Best Case Feature Selection



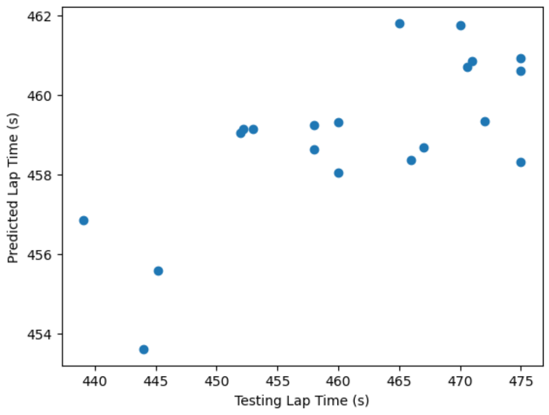
(e) Predicted vs Testing Times Using Gradient Boosted Trees With Grid Search Cross Validation

Figure 2: Summary of Graphs for Gradient Boosted Trees Algorithm

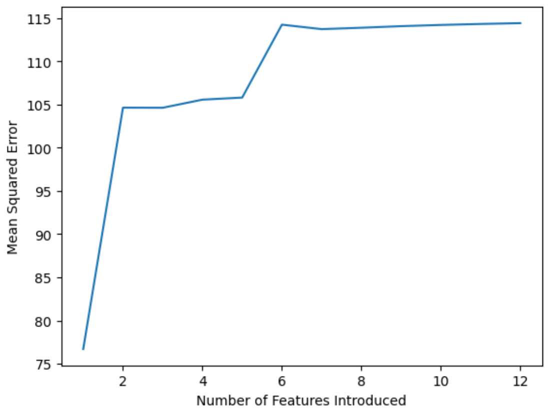
Appendix C Support Vector Regression Graphs



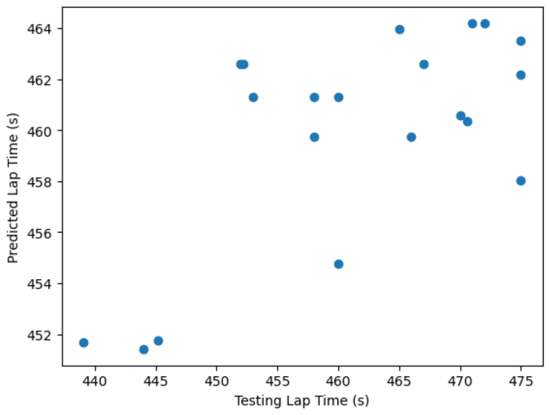
(a) Predicted vs Testing Times Using Support Vector Regression (Base Case)



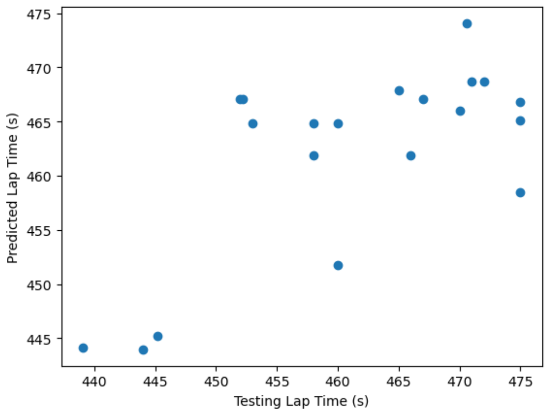
(b) Predicted vs Testing Times Using Support Vector Regression (Post Scaling)



(c) How Number of Features Affects Mean Squared Error for Support Vector Regression



(d) Predicted vs Testing Times Using Support Vector Regression Using Best Case Feature Selection



(e) Predicted vs Testing Times Using Support Vector Regression With Grid Search Cross Validation

Figure 3: Summary of Graphs for Support Vector Regression Algorithm