Predicting the Road Surface Quality Score using Regression Models

Sindhuri Punyamurthula
San Diego State University
spunyamurthula@sdsu.edu

Abstract

This paper proposes regression models to predict the road surface score on a scale of 0 to 3 with 0 for unworn roads up to 3 for heavily worn roads. Various regression models have been used for the prediction. The dataset used is KITTI image dataset and each image is converted to a histogram of color with different bin sizes (10,16) used as feature vector for various models. Linear Regression, Support Vector Regression, Neural Networks, Random Forest Regressors have been used for predictions. Finally, we compare the performance of each of these model in terms of Root Mean Square Error and R2 score to see which model is the best in predicting the score.

1. Introduction

The main motivation behind this project is to provide an image based road quality information. The maintenance of road quality information is a challenging task as road conditions keep changing due to various factors like weather, traffic load and normal wear and tear over a certain period of time. In the future, this information can be used by drivers to estimate the speed at which they can drive thereby avoiding hazardous situations. The primary focus of this project is to work towards finding the best approach to predict road quality score from simple images of the road surface. I have experimented with various regression models and compared the quantitative results of each model to determine the best model producing the least errors.

2. Dataset Information

KITTI image dataset was used for this model. The images were passed manually through a Java annotator which created a csv file with the image name and the score associated of the road in the image. A score of -1 indicated unrelated data, 0 indicated unworn 1 indicated lightly worn, 2 was for worn road and 3 was for heavily worn road.

A python program was written to capture the histogram of color of each image in the previously created csv file. A two-dimensional matrix of size (125,414) was created after converting the image to grayscale. This was flattened to a 1-dimensional array. So, the image was first converted to grayscale and then histogram was created using bins size of 10 and 16. The histogram information of the images was stored in two separate csv files for each bin size. Both csv files were used as the dataset for the models separately to determine the optimal bin size that made our models perform better.

3. Environment setup

Jupyter Ipython notebook and Python’s sci-kit package was used for applying models. Matplotlib was used to work with images and plot the graphs during model evaluation. It helped in visualization of the results and data. Pandas was used for data manipulation and analysis. The Pandas data frames ensured high-performance of the algorithms. Numpy, a fundamental library was used for scientific computing. Many other python libraries were also used. More information about the project files is provided at the end of the report.

4. Data Pre-processing

Data Pre-processing is a way of organizing selected data by formatting, cleaning and sampling from it. More data can result in larger computational and memory requirements. Feature selection is the process of selecting relevant features that can be applied to construct our model and helps in improving the model performance.

We have achieved by considering the performance of our model with different bin sizes of the histogram of color (10 and 16).

Feature scaling or standardization ensures that all our data rows and columns lie within the same range instead of a mix up of varying numbers. MinMaxScaler with range (0 to 1) and StandardScaler were used in the preprocessing phase.

3.1. Histogram of image data

Firstly, we read an image from a file into an array. For grayscale images, the return array is MxN. For RGB images, the return value is MxNx3. Then we calculate the histogram, which is a graph showing the number of pixels in an image at each different intensity value found in that image. We used two bin sizes 10 and 16 in this paper.

This is how an image histogram looks like. The following is a histogram image with 10 bins.

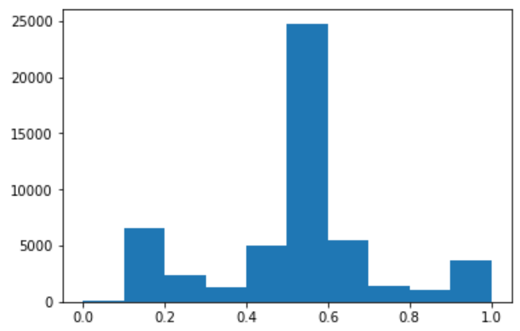
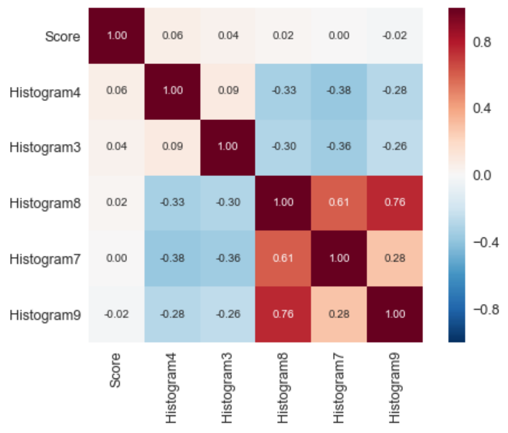


Figure 1. Histogram of image (10 bins)

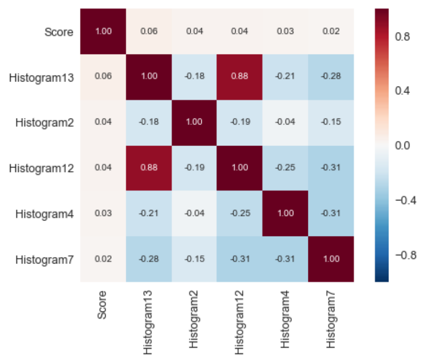
3.2. Correlation matrix

A correlation matrix is a table showing correlation coefficients between sets of variables. Each random variable (X_i) in the table is correlated with each of the other values in the table (X_j). This allows you to see which pairs have the highest correlation.



```
['Histogram4', 'Histogram3', 'Histogram8', 'Histogram7', 'Histogram9
```

Figure 2. Correlation Matrix (10 bins)



```
['Histogram13', 'Histogram2', 'Histogram12', 'Histogram4', 'Histogram7']
```

Figure 3. Correlation Matrix (16 bins)

The correlation matrix gives the top 5 highly correlated features in both 10 bins and 16 bin files. Using these features as input for various models, gives similar results as using the whole 10 or 16 feature set. Hence all 10 and 16 columns have been used as input data for all the models.

First, the models and their results are compared with 10 bin datasets. Next, the model with 16 bin datasets is created and errors are compared.

4. Machine Learning Models

In this section, we discuss about various models that that been applied and their performance. Cross-validation is a process for reliably estimating the performance of a method for building a model by training and evaluating the model multiple times using the same method.

In this project, GridSearchCV was used which essentially performs cross-validation across an entire grid (all possible permutations) of hyperparameters.

A cross validation pipeline model is used in this paper which includes preprocessing, specifying the hyperparameters to tune and the number of folds to create.

4.1. Linear Regression

Python scikit’s sklearn. linear_models was used for Linear Regression was used here. Linear Regression is a linear model that assumes a linear relationship between input variables (X_train) and single target/output variable y_train. Simple linear regression is when there is a single input variable, while with multiple input variables, the method is called multiple linear regression. Both the input and output values are numeric and continuous.

$$y = f_\theta(x) = \theta_0 + \theta_1\, x^1 + \theta_2\, x^2 + \theta_3 x^3 + \dots$$

$$\theta\ is\ weights\ , x\ are\ input\ features$$

$\theta = [\theta_0,\ \theta_1,\ \theta_2, \theta_3 \dots],\ \ x = [1,\ x^1,\ x^2, x^3, \dots]$ then

$$y = f_\theta(x) = \theta^T x$$

Let $x_i = [1,\ x_i^1, x_i^2, x_i^3]$ represent the i-th training sample, y_i semantic label.

The goal is to minimize the cost function

$$F(\theta) = \frac{1}{2m}\sum_i \big(y_i - f_\theta(x_i)\big)^2$$

The Ordinary Least Squares procedure seeks to minimize the sum of the squared residuals. This means that given a regression line through the data we calculate the distance from each data point to the regression line, square it, and sum all the squared errors together.
Two types of regularization procedures are: -
i. Lasso Regression: where Ordinary Least Squares is modified to also minimize the absolute sum of the coefficients (called L1 regularization).
ii. Ridge Regression: where Ordinary Least Squares is modified to also minimize the squared absolute sum of the coefficients (called L2 regularization).
These methods are effective to use when there is collinearity in your input values and ordinary least squares would overfit the training data.

4.1.1. Hyperparameters used for Linear Regression Model

We perform multiple linear regression of the histogram of image with different bin sizes. The model which uses 16 bin size histogram data performs better than the 10-bin data.

The different hyperparameters used in this Linear Regression model were
1) fit_intercept: [True, False]
This is used to specify whether to calculate the intercept for this model. If set to False, no intercept will be used in calculations (e.g. data is expected to be already centered).
2) normalize: [True, False]
This parameter is ignored when fit_intercept is set to False. If True, the regressors X will be normalized before regression by subtracting the mean and dividing by the l2-norm.

3) copy_X: [True, False]
If True, X will be copied; else, it may be overwritten.

The best parameters that fit the model with 10 bins is {'linearregression__copy_X': True, 'linearregression__fit_intercept': False, 'linearregression__normalize': True}

The best parameters for 16 bins are {'linearregression__copy_X': True, 'linearregression__fit_intercept': False, 'linearregression__normalize': True}

4.2. Support Vector Regression

Python sklearn.svm SVR () was used in this project for Support Vector Regression.SVM regression performs linear regression in the high-dimension feature space using ε-insensitive loss and, at the same time, tries to reduce model complexity by minimizing $\|\omega\|^2$. This can be described by introducing (non-negative) slack variables $\xi_i, \xi_i^*\ \ i = 1,\dots n$, to measure the deviation of training samples outside ε-insensitive zone. Thus, SVM regression is formulated as minimization of the following function:

Minimize

$$\frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)$$

s.t.

$$\begin{cases} y_i - f(\mathbf{x}_i, \omega) \le \varepsilon + \xi_i^* \\ f(\mathbf{x}_i, \omega) - y_i \le \varepsilon + \xi_i \\ \xi_i, \xi_i^* \ge 0, i = 1,\dots,n \end{cases}$$

4.2.1. Hyperparameters used for SVR Model

The Support Vector Regression model was used to predict the scores of both files containing histogram of image data with 10 and 16 bins.
Different hyperparameters that were used to tune the model are: -
1) C: [0.001, 0.01, 0.1, 1]
Which is penalty parameter C of the error term.

2) gamma: [0.001, 0.01, 0.1, 1]

Kernel coefficient for 'rbf', 'poly' and 'sigmoid'. If gamma is 'auto' then 1/n_features will be used instead.

3) kernel: ['linear','poly','rbf','sigmoid']
Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to precompute the kernel matrix.

The best parameters for 10 bins are
{'svr__C': 0.1, 'svr__gamma': 1, 'svr__kernel': 'rbf'}

The best parameters for 16 bins are
{'svr__C': 0.01, 'svr__gamma': 1, 'svr__kernel': 'rbf'}

4.2.2. Effect of kernel type on the RMSE

a) 10 bins data

Considering C=0.1 and gamma=1, i.e. the best parameters the RSME values seems to be the highest for sigmoid kernel.

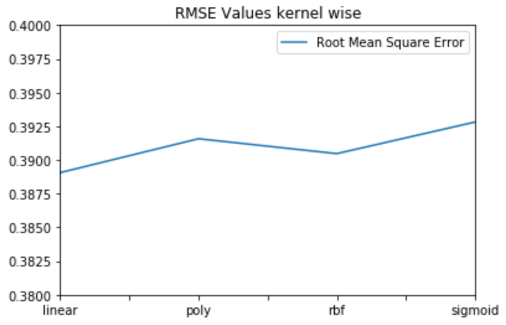


Figure 4. RMSE using different kernels (10 bins)

b) 16 bins

Considering C=0.01 and gamma=1, i.e the best parameters the RSME values seems to be the highest for sigmoid kernel and lowest for poly kernel.

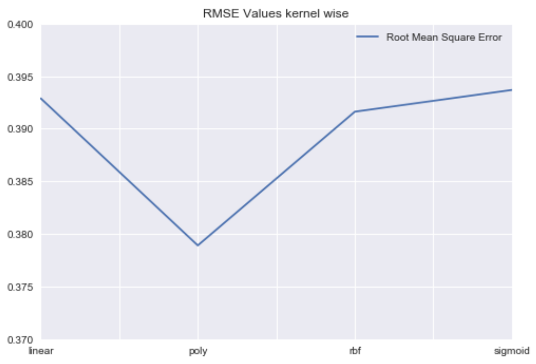


Figure 5. RMSE using different kernels (16 bins)

4.2.3. Effect of C on the RMSE

Taking different C values for Support Vector Regression, we can say that a small C value like 0.0001 gives a high Root Mean square error.C=0.1 is the optimal value giving the lowest RMSE.This applies to both 10 and 16 bin data.

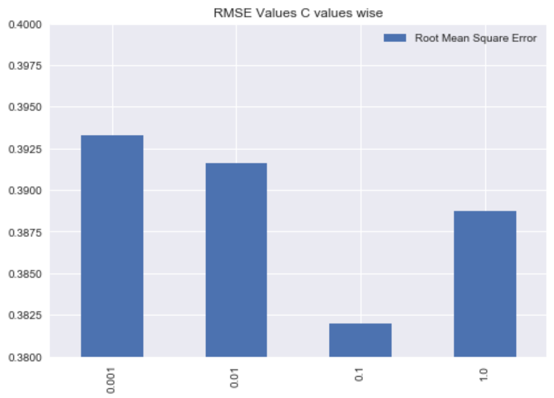


Figure 6. RMSE using C values

4.2.3. Effect of gamma on the RMSE

Here a sigmoid kernel type with C value 1, is taken to observe the effect of gamma values on RMSE.All others values seem to have the same effect on RMSE except when gamma=1, where the error increases. These patterns can be seen in both 10 and 16 bins input features.

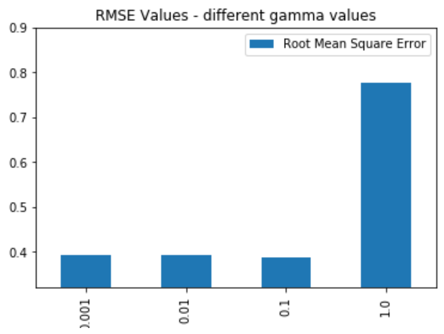


Figure 7. RMSE using gamma values

4.3. Regression using Neural Networks

Python scikit's MLPRregressor was used in this project. Class MLPRegressor implements a multi-layer perceptron (MLP) that trains using backpropagation with no activation function in the output layer, which can also be seen as using the identity function as activation function. Therefore, it uses the square error as the loss function, and the output is a set of continuous values.

4.3.1. Hyperparameters used for Neural Network Regression Model

The different hyperparameters that were used to tune the Neural Network Regression models are

1) hidden_layer_sizes: [3,4,5,8,10,12,20,30,50,80]
2) activation: ['identity','tanh','relu']
Activation function for the hidden layer.

- 'identity', no-op activation, useful to implement linear bottleneck, returns f(x) = x
- 'logistic', the logistic sigmoid function, returns f(x) = 1 / (1 + exp(-x)).
- 'tanh', the hyperbolic tan function, returns f(x) = tanh(x).
- 'relu', the rectified linear unit function, returns f(x) = max (0, x)

3) solver: ['lbfgs','sgd','adam']
The solver is for weight optimization.

- 'lbfgs' is an optimizer in the family of quasi-Newton methods.
- 'sgd' refers to stochastic gradient descent.
- 'adam' refers to a stochastic gradient-based optimizer proposed by Kingma, Diederik, and Jimmy Ba

The best parameters for 10 bin data are: {'mlpregressor__activation': relu, 'mlpregressor__hidden_layer_sizes': 80, 'mlpregressor__solver': 'adam'}

The best parameters for 16 bin data are: {'mlpregressor__activation': relu, 'mlpregressor__hidden_layer_sizes': 4, 'mlpregressor__solver': sgd}

4.3.2. Effect of hidden layer size on the RMSE

For both 10 and 16 bin data the root mean square error is lowest if the value of hidden layer size is 50. We have taken relu activation function and adam solver. With 10 bins, neural networks with hidden layer size 50 or 80 seems to be the best.

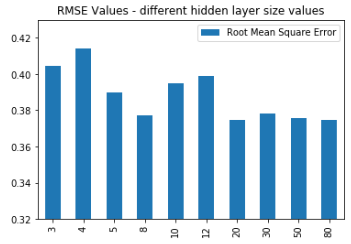


Figure 8. RMSE using hidden layer size values (10 bin)

While with 16 bins, hidden layer size of 50 works best.

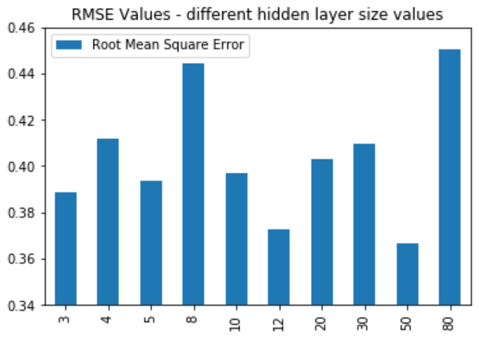


Figure 9. RMSE using hidden layer size values (16 bins)

4.3.3. Effect of solver on the RMSE

The most optimal solver for both models is adam when default values of MLPRegressor are considered.

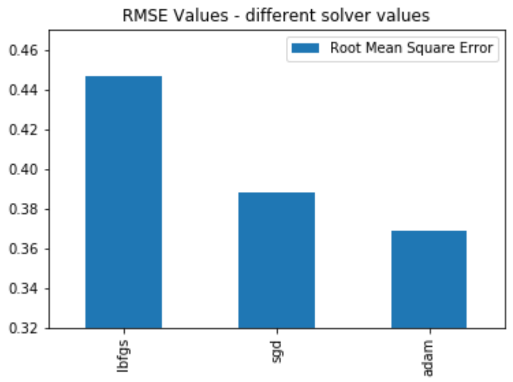


Figure 10. RMSE using solver values

4.3.4. Effect of activation function on the RMSE

For both the 10 and 16 bins, relu function works best when default values of MLPRegressor are considered.

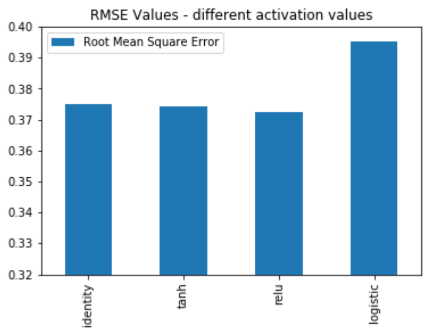


Figure 11. RMSE using different activation function

4.3.5. Effect of other parameters on the RMSE

The model performs similarly in case we use different alpha values [0.0001,0.001,0.01,1] for 10 bin data. However alpha=1 works best for 16-bin data.
With epsilon values [1e-08,1e-02,1e-05,1e-03], the rmse is lowest when epsilon is 1e-08 for both models.

4.3. Additional models used

Bayesian Ridge, Huber Regressor, Bagging, Random Forest Regression, AdaBoostRegressor are some of the other models which have been used in the paper. The primary focus was on Support Vector Regression, Linear Regression and Neural Network Regression.

5. Performance Evaluation

In this section we have experimented with various regression models and we compare their performance based on two evaluation metrics Root Mean Square Error and Mean Absolute Error.
The RMSE is the square root of the variance of the residuals. It indicates the absolute fit of the model to the data–how close the observed data points are to the model's predicted values. Lower values of RMSE indicate better fit.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

MAE measures the average magnitude of the errors in a set of predictions, without considering their direction.

$$\mathrm{MAE} = \frac{1}{n}\sum_{j=1}^{n}|y_j - \hat{y}_j|$$

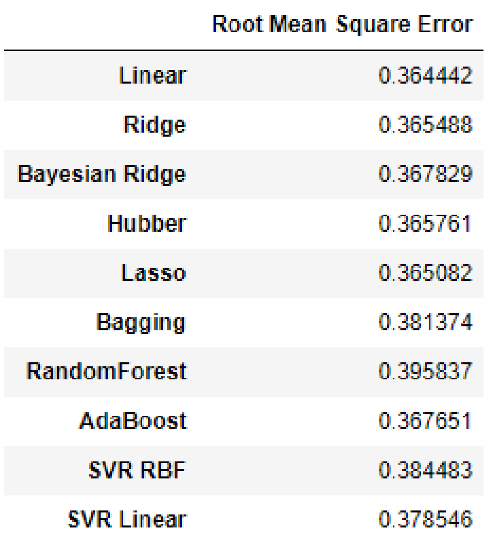
A comparison was made between the three regression models SVR, Linear and Neural Networks using both 10 bins and 16 bin data. The Neural network model is the best model with the least Root Mean Square Error.

Model	10 bins data	16 bins data
Linear	0.3705	0.3667
SVR	0.3904	0.3916
Neural Network	0.3742	0.3660

Table 1. RMSE using 3 different models

In another additional work, an experiment was performed using various regression models with K-Fold Cross-Validation technique. Their RMSE and MAE were compared to determine which performs the best.

5.1. Comparison of RMSE for additional models



	Root Mean Square Error
Linear	0.364442
Ridge	0.365488
Bayesian Ridge	0.367829
Hubber	0.365761
Lasso	0.365082
Bagging	0.381374
RandomForest	0.395837
AdaBoost	0.367651
SVR RBF	0.384483
SVR Linear	0.378546

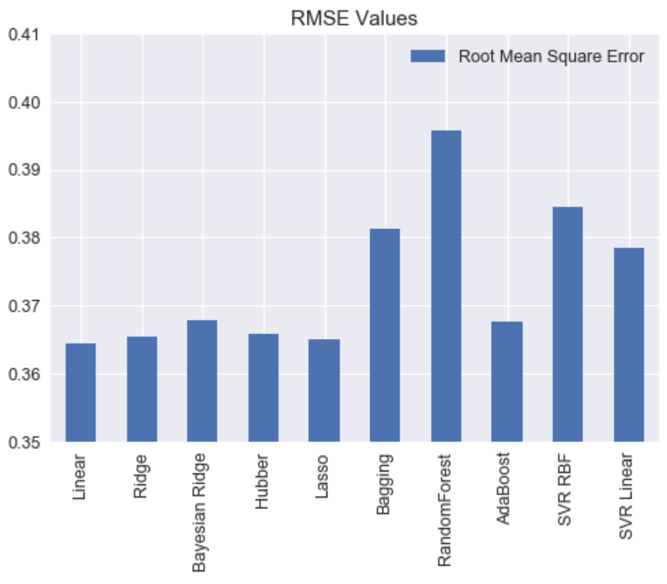
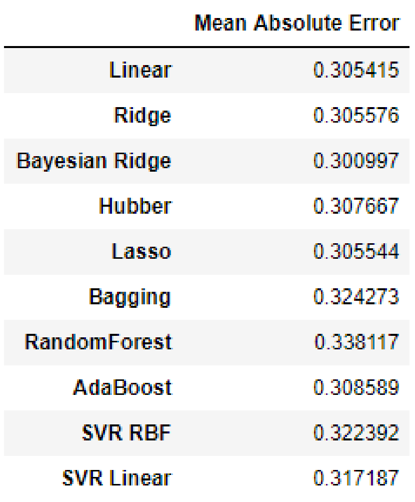


Figure 12. RMSE using different models

According to the table and figure, Linear, Lasso and AdaBoostRegressor are the best performing models due to their least root mean square error. Random Forest and Bagging Regressor do not perform well as the error value is very high.

5.2. Comparison of MAE for additional models



	Mean Absolute Error
Linear	0.305415
Ridge	0.305576
Bayesian Ridge	0.300997
Hubber	0.307667
Lasso	0.305544
Bagging	0.324273
RandomForest	0.338117
AdaBoost	0.308589
SVR RBF	0.322392
SVR Linear	0.317187

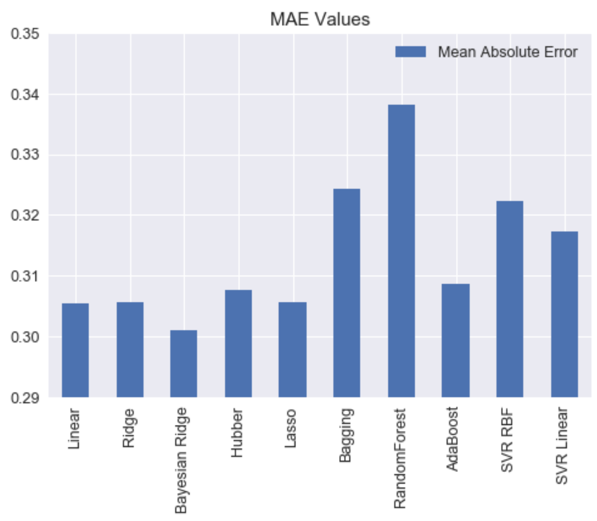


Figure 13. RMSE using different models

According to the figure, Linear, Lasso and Bayesian Ridge Regressor are the best performing models due to their mean absolute error. Random Forest and Bagging Regressor do not perform well as the error value is very high.

6. Conclusion

In this project, various regression models were applied to predict the road surface score. We used histogram of image of different bin sizes as input features. The MinmaxScaler () was applied in the preprocessing stage. GridSearchCV and K-fold cross validation techniques were used to avoid overfitting and under fitting. Neural Network model on 16 bins data was the best model with the least root mean squared error of 36% and mean absolute error of 31 %.
Linear, Lasso, AdaBoost Regressor also have similar and equally good rmse values.

7. Project Information

This section contains information about the location of code file, datasets and other related information.

7.1 Input Files

There are two input files:

1) **./ML_ANNOTATOR/HistScore.csv for 10 bins**
2) **./ML_ANNOTATOR/hist16bin.csv for 16bins**

These input files contain the already extracted histogram of image data using 10 bins and 16 bins respectively.
The column names Histogram1, Histogram2.... i.e. Histogrami indicate the number of pixels in the ith bin.

7.2 Code File

Name: Sindhuri_ML_Project_Code.ipynb

The input data is used in various models. Running the code directly in the notebook in order can show the desired outputs.

7.3 Additional Information

The code for extracting the histogram of image into different bin sizes of 10 and 16 are shown in
i) CreateHist(10bins). ipynb
ii) CreateHist(16bins). ipynb
README.txt file has more detailed steps about starting from the scratch working with the original image dataset.

8. References

[1] Scikit-Learn Package Python - http://scikitlearn.org/stable/

[2] http://www.cvlibs.net/datasets/kitti/

[3] https://www.kaggle.com/

[4] Prof. Xiaobai Liu, PhD, SDSU (Department of Computer Science) – CS596 Lecture slides

[5] Prof. Shawn Healy, SDSU provided with Annotator tool for creating the csv file.

[6] https://machinelearningmastery.com/metrics-evaluate-machine-learning-algorithms-python/

[7] https://machinelearningmastery.com/prepare-data-machine-learning-python-scikit-learn/