Final Report

Predicting rock density from X-Ray frequencies with Machine Learning

Nibin Koshy
McMaster University
Course: SEP787 Machine Learning: Classification Models
Dr. Sayyed Faridoddin Afzali
Date: August 06, 2024

1.Problem Explanation:

The work tries to understand the corelation between X-Ray signals and rock densities for a tunnel boring company. This will allow the boring company to switch out the boring heads on their equipment before having to mine through the rocks. The dataset contains the frequency from the X-ray equipment and a corresponding rock density from lab tests. The signal strength is noted in nHz while the rock density is noted in kg/m^3. By using a suitable machine learning technique, we can try to predict the rock density from the X-Ray signals with good accuracy and thereby aid the company in selecting a suitable boring head for their equipment to carry out their operations.

2.Data preprocessing steps:

After loading the csv file, we view the DataFrame using the head function. We then rename the columns from the existing names of "Rebound Signal Strength nHz" and "Rock Density kg/m3" to "signal" and "density" for simplicity. Here, signal is the single feature and density is the target variable.

In order to make the model compatible with scikit learn, we need to reshape the 'signal' column. By reshaping the array using reshape(-1, 1), we transform it into a 2-dimensional array with the shape (n_samples, 1), where the second dimension indicates that there is only one feature:

- -1 in reshape tells numpy to automatically determine the number of rows based on the length of the array and the specified number of columns.
- 1 specifies that there is one feature.

We then use the train test split method from the sklearn.modelselection library to split the dataset into a test set and a train set. We set apart 10% of the data for testing with a fixed random state fixed.

3.Methods performance comparison

To compare multiple methods, we define a function called "run_model" that trains a given machine learning model, evaluates its performance, and visualizes the results. We calculate the root mean squared error (RMSE) between the true target values (y_test) and the predicted values (preds).

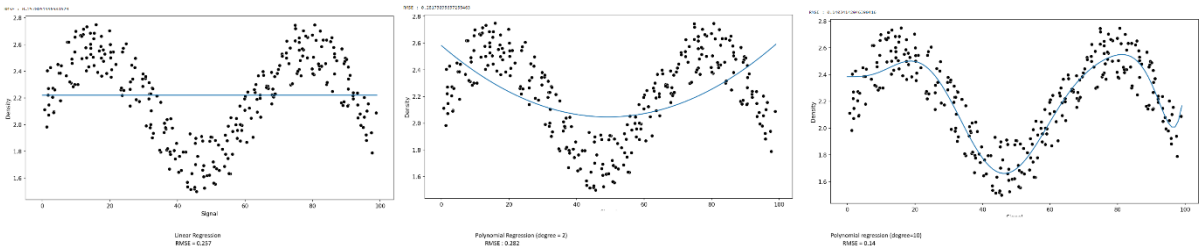
We also visualise the results using a scatterplot containing the existing values and plot a line plot with predicted values of 'signal' ranging from 0 to 99. The methods used in the project are

1. Linear Regression
2. Polynomial Regression
3. KNN Regression
4. Decision Tree Regression
5. Support Vector Regression
6. Random Forest Regression
7. Gradient Boost Regression
8. AdaBoost Regression
9. Random Forest with cross validation regression

3.1 Linear and polynomial regression

Linear regression in scikit-learn is a straightforward model that assumes a linear relationship between the input features and the target variable. It fits a line (or hyperplane in higher dimensions) to the data that minimizes the sum of squared differences between the observed and predicted values. Polynomial regression, on the other hand, extends linear regression by incorporating polynomial features of the input data, allowing for more complex, non-linear relationships. In scikit-learn, this can be achieved using `PolynomialFeatures` to generate polynomial and interaction terms, followed by fitting a linear regression model on the transformed features. This combination allows polynomial regression to capture more intricate patterns in the data while maintaining the simplicity of linear regression for modeling. We try polynomial regression with a degree of 2 and a degree of 10 (Figure 1).

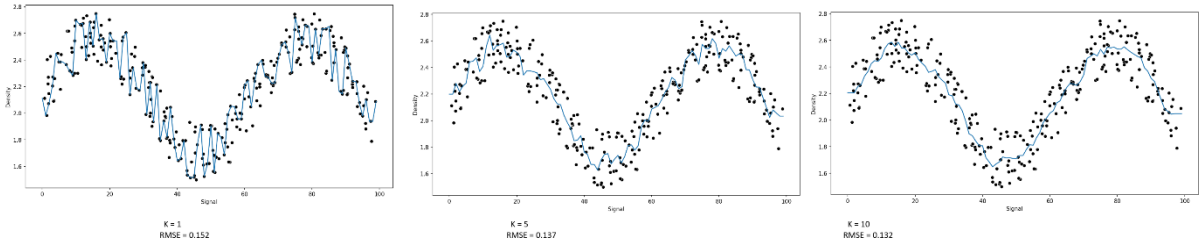
Fig 1. Visualization between linear and polynomial regression



3.2 KNN Regression

K-Nearest Neighbors (KNN) regression in scikit-learn is a non-parametric method that predicts the target value for a given input by averaging the target values of its 'k' nearest neighbors in the training data. For k = 1, the model simply takes the target value of the closest training data point, making it highly sensitive to noise. When k = 5, the prediction is the average of the five nearest neighbors, offering a balance between capturing local variations and smoothing out noise. Increasing k to 10 further smoothens the predictions, as it considers a broader neighborhood, which can reduce the model's sensitivity to outliers but may also overlook finer patterns. Each k value provides a different trade-off between bias and variance, impacting the model's performance based on the specific data characteristics (Figure 2).

Fig 2. Visualization between KNN Regressions



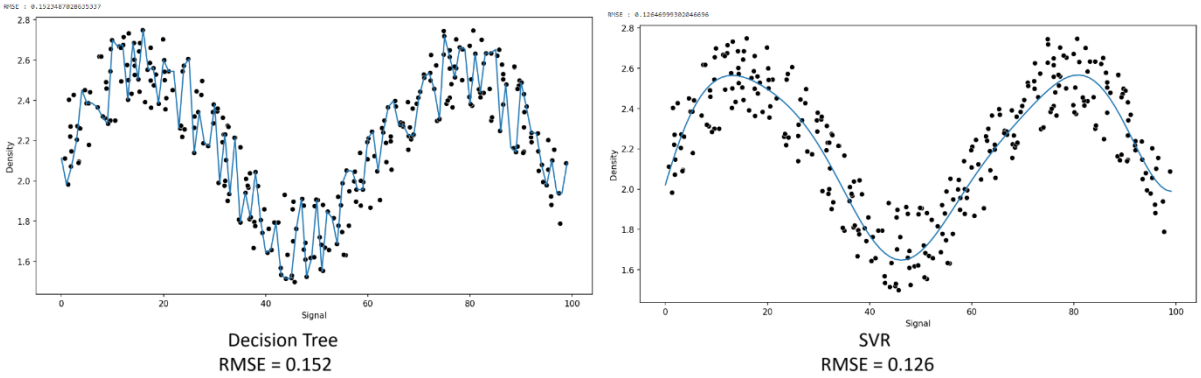
3.3 Decision Tree Regression and Support Vector Regression

Decision Tree Regression in scikit-learn involves splitting the data into subsets based on the feature values that result in the most homogeneous subsets, according to a criterion like mean squared error. This model builds a tree where each leaf represents a predicted target value. It's intuitive and can capture non-linear relationships, but it can overfit the training data without proper pruning. Support Vector Regression (SVR) aims to find a hyperplane in a high-dimensional space that best fits the data, using a margin of tolerance (epsilon) to handle deviations. It employs

kernel functions to model non-linear relationships, offering robustness to outliers and flexibility in capturing complex patterns. Both methods are powerful, with Decision Tree Regression providing interpretability and SVR offering robust, non-linear modelling capabilities.

Decision Tree Regression and Support Vector Regression (SVR) both offer flexibility and highlight important features, with Decision Trees using hierarchical splits and SVR employing kernel functions. However, they differ significantly: Decision Trees are highly interpretable through their tree structure, but can easily be overfit without pruning, while SVR models are less interpretable but include regularization to balance model complexity and error tolerance. Decision Trees are sensitive to data changes and typically faster to train, whereas SVR is more robust to outliers and computationally intensive, especially with non-linear kernels (Figure 3).

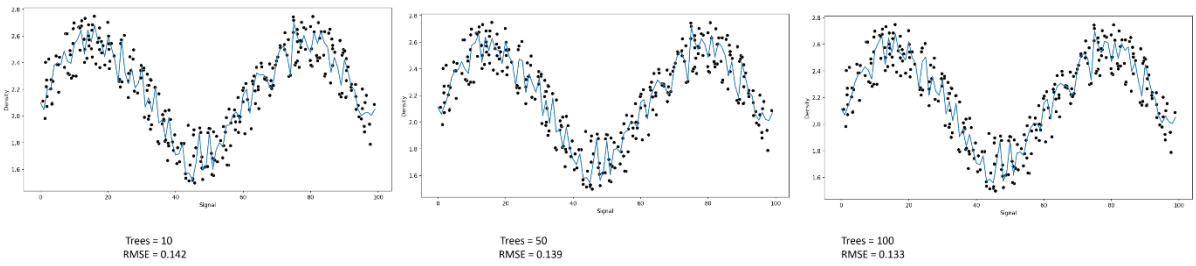
Fig 3. Visualization between decision tree and SVR.



3.4 Random Forest Regression

Random Forest Regression in scikit-learn is an ensemble method that builds multiple decision trees and averages their predictions for more accurate and stable results. The `n_estimators` parameter controls the number of trees in the forest. With `n_estimators = 10`, the model is quick to train but may not fully capture the data's complexity, leading to higher variance. Increasing to `n_estimators = 50` offers a better balance, reducing variance and improving predictions by combining more diverse trees. At `n_estimators = 100`, the model further stabilizes, achieving higher accuracy and robustness, though with increased computational cost and training time. Each increment in `n_estimators` enhances the model's performance and reduces overfitting by averaging more predictions (figure 4).

Fig 4. Visualization between random forest regression with different tree sizes

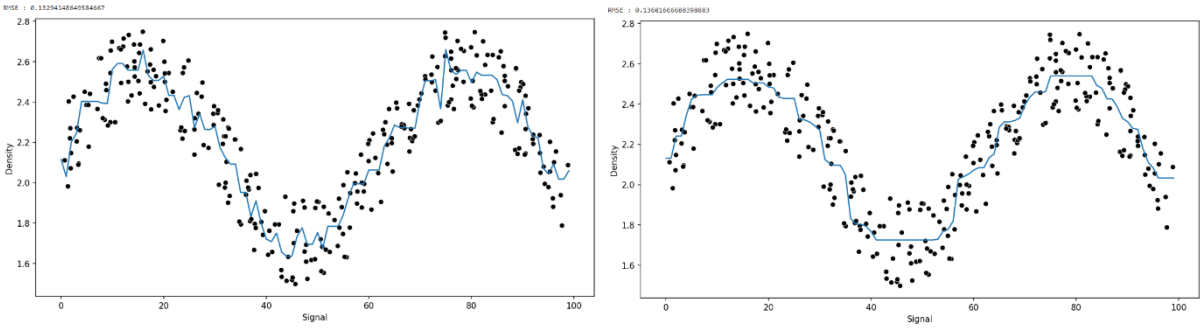


3.5 Gradient boost and AdaBoost Regression

Gradient Boosting and AdaBoost are both ensemble methods that enhance predictive performance by combining multiple weak learners, typically decision trees, into a single strong model. Gradient Boosting builds models sequentially, where each new model corrects the errors

of the previous ones by minimizing a loss function, leading to high accuracy and robustness against overfitting. AdaBoost, short for Adaptive Boosting, also builds models sequentially but focuses on correcting errors by giving more weight to misclassified instances, improving performance on difficult examples. While Gradient Boosting is often more flexible and powerful due to its gradient-based optimization, AdaBoost is simpler and can be more interpretable but might be more sensitive to noisy data. Both methods effectively reduce bias and variance, though they do so with different strategies and complexities (Figure 5).

Fig 5. Visualization between gradient boost and adaboost regression.



Gradient Boosting
RMSE : 0.133

AdaBoost
RMSE : 0.134

3.6 Random forest with grid search and cross validation

Random Forest Regression is a versatile ensemble learning technique that combines multiple decision trees to improve predictive accuracy and control overfitting. It builds a multitude of trees during training, each trained on a random subset of the data and features, and then aggregates their predictions to provide a robust final output. To optimize the performance of a Random Forest Regressor, Grid Search and Cross-Validation are commonly employed. Grid Search systematically explores a predefined set of hyperparameters, such as the number of estimators, maximum depth of the trees, and minimum samples per leaf, to find the combination that yields the best model performance. Cross-Validation is used to assess how well the model generalizes to unseen data by splitting the dataset into multiple folds and training the model on various training subsets while evaluating it on the remaining fold. This process helps in estimating the model's performance more reliably and reducing the risk of overfitting. By integrating Grid Search with Cross-Validation, one can efficiently fine-tune the hyperparameters of the Random Forest Regressor, ensuring that the model achieves optimal performance and robustness across different subsets of the data, leading to better predictive accuracy and generalization.

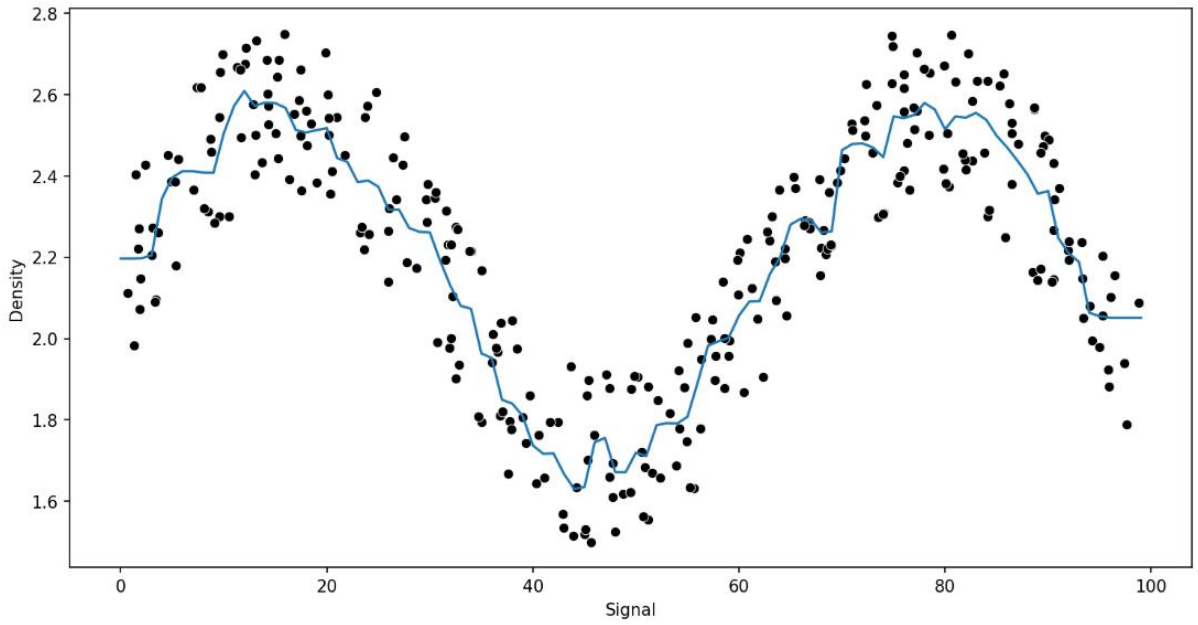
Using Grid search it was found that the best hyperparameters for random forest were:

{'bootstrap': True, 'max_depth': None, 'max_features': 'auto', 'min_samples_leaf': 4, 'min_samples_split': 10, 'n_estimators': 100}

The parameters are as follows: `'bootstrap': True` indicates that bootstrap sampling is used to train each tree on a random subset of the data. `'max_depth': None` means that the trees are allowed to grow until all leaves are pure or contain fewer than the minimum samples required. `'max_features': 'auto'` specifies that the number of features considered for splitting a node is automatically determined, typically the square root of the total number of features. `'min_samples_leaf': 4` ensures that each leaf node contains at least 4 samples, which helps in controlling overfitting. `'min_samples_split': 10` requires at least 10 samples in a node before it

can be split, further preventing overfitting. `'n_estimators': 100` sets the number of decision trees in the forest to 100, balancing model complexity and computational efficiency. These settings collectively aim to optimize the model's performance and generalization by carefully controlling tree growth and complexity. The visualization for this method can be seen in figure 6.

Fig 6. Visualization for random forest with best hyperparameters



RMSE : 0.137

4.Results

The RMSE for the different methods employed can be seen in table 1.

Method	Linear	Polynomial		KNN			Dec. Tree	SVR	Random Forest	Gradient Boost	Ada Boost	Random forest with best parameters
		2 deg.	10 deg.	K = 1	K = 5	K = 10						
RMSE	0.257	0.282	0.14	0.152	0.137	0.132	0.152	0.126	0.142	0.133	0.134	0.137

Table 1: RMSE values for different machine learning methods

The table presents the Root Mean Squared Error (RMSE) for various regression methods and configurations, providing insight into their predictive performance. Linear regression shows an RMSE of 0.257, while polynomial regression with a 2-degree polynomial increases RMSE to 0.282, and a 10-degree polynomial decreases RMSE to 0.14. K-Nearest Neighbours (KNN) performs well with K = 1 at 0.152, K = 5 at 0.137, and K = 10 at 0.132, demonstrating the impact of different K values on performance. Decision Tree Regression has an RMSE of 0.152, and Support Vector Regression (SVR) achieves the lowest RMSE of 0.126. Random Forest Regression gives an RMSE 0.142. Gradient Boosting and AdaBoost have RMSE values of 0.133 and 0.134, respectively, with Gradient Boost performing slightly better. Random Forest with optimized parameters achieves an RMSE of 0.137, indicating effective hyperparameter tuning but still not as low as SVR. This comparison illustrates how different models and configurations affect prediction accuracy.

5. Conclusion

In conclusion, the results reveal that Support Vector Regression (SVR) delivers the best predictive performance with the lowest RMSE of 0.126, highlighting its effectiveness in capturing the underlying data patterns. Among other models, Polynomial Regression with a 10-degree polynomial and K-Nearest Neighbours with K = 10 also show strong performance with low RMSE values of 0.140 and 0.132, respectively. Decision Tree Regression and Random Forest Regression, while providing reasonably low RMSEs, do not match the accuracy of SVR. Gradient Boosting and AdaBoost exhibit competitive performance, with Gradient Boosting slightly outperforming AdaBoosting. Random Forest with optimized parameters achieves an RMSE of 0.133, demonstrating the benefits of hyperparameter tuning.

Overall, it would be advisable to use SVR or KNN method with K=10 for predicting the rock density from the X-Ray frequency and selecting the appropriate head for the tunnel boring equipment.