DS 203: Programming for Data Science - Exercise 3

Swayam Saroj Patel (22B1816)

February 16, 2025

Review of Jupyter Notebook:

Part a (Summary):

The code here starts of by importing all of the libraries needed. Following which it is divided into the test and train division. Then the data is scaled using *StandardScaler()*. We then see an application of feature engineering, and a csv file of it is created. Then the feature engineered data is trained and tested. The algorithms used are:

- Linear Regression
- SVM Regression
- Random Forest Regression
- Gradient Boosting Regression
- k-NN Regression
- Neural Network Regression

And the model evalution metrics used are:

- Mean Squared Error (MSE)
- R-squared
- Durbin-Watson Statistic
- Jarque-Bera Test

And lastly the graphs are plotted and the metrics values are calculated showing which model performs well in this data set.

Part b(Learning):

One of the main thing that I would like to highlight is the use of *StandardScaler()*. It made me ask, why? And into the internet I go and got to know a bit about it. So the reason why Standardization is needed is because of algorithms like SVM and K-NN clustering that are highly depended on the distances for evaluation and execution of the model.

SkLearn Function documentation table:

- **PolynomialFeatures(degree)**:
  - Generates polynomial and interaction features up to a specified degree.
  - Expands the feature set to capture non-linearity in the data.
  - Parameters:
    - `degree` (int): The degree of the polynomial features.
    - `interaction_only` (bool, default=False): If True, only interaction features are produced.
    - `include_bias` (bool, default=True): If True, includes a bias column (all ones).
  - Output: Transformed feature matrix with polynomial combinations of input features.
- **StandardScaler()**:
  - Standardizes features by removing the mean and scaling to unit variance.
  - Parameters:
    - `with_mean` (bool, default=True): Centers the data before scaling.
    - `with_std` (bool, default=True): Scales data to unit variance.
  - Output: Scaled feature matrix with zero mean and unit variance.
- **LinearRegression()**:
  - Implements Ordinary Least Squares (OLS) regression to model linear relationships between input features and target variables.
  - Parameters:
    - `fit_intercept` (bool, default=True): Whether to calculate the intercept for this model.
    - `normalize` (bool, default=False): Whether to normalize input features before fitting.
  - Output: Fitted regression model with coefficients and intercept.
- **SVR(kernel='poly')**:
  - Support Vector Regression using a polynomial kernel to capture non-linearity in data.
  - Parameters:
    - Common parameters include: - Kernel type (`'poly'` for polynomial). - Regularization parameter (`C`, default=1.0). - Degree of the polynomial kernel (`degree`, default=3).
  - Output: Fitted SVR model for regression tasks.
- **RandomForestRegressor()**:
  - An ensemble learning method using multiple decision trees. It is robust to overfitting and works well with non-linear data.

- Parameters:
  - Number of trees (`n_estimators`, default=100). - Maximum depth of trees (`max_depth`, optional). - Criterion for split quality (`'mse'` or others).
- Output: Fitted random forest regression model.

- **GradientBoostingRegressor()**:
  - A boosting technique that builds trees sequentially to minimize errors. It combines weak learners to create a strong learner.
  - Parameters:
    - `n_estimators` (int, default=100): Number of boosting stages to perform.
    - `learning_rate` (float, default=0.1): Shrinks the contribution of each tree.
    - `max_depth` (int, default=3): Maximum depth of the individual regression estimators.
  - Output: Fitted Gradient Boosting model for regression tasks.

- **MLPRegressor(hidden_layer_sizes, max_iter)**:
  - Implements a Multi-layer Perceptron (Neural Network) for regression tasks.
  - Parameters:
    - `hidden_layer_sizes` (tuple, default=(100,)): Number of neurons in each hidden layer.
    - `max_iter` (int, default=200): Maximum number of iterations for optimization.
    - Other parameters include activation function (`'relu'`), solver (`'adam'`), and learning rate.
  - Output: Fitted neural network regression model.

- **KNeighborsRegressor()**:
  - A non-parametric algorithm that predicts based on the k-nearest training samples.
  - Parameters:
    - `n_neighbors` (int, default=5): Number of neighbors to use for prediction.
    - Distance metric (`'minkowski'`, default) and weights (`'uniform'` or 'distance').
  - Output: Predictions based on the nearest neighbors.

- **mean_squared_error(y_true, y_pred)**:
  - Computes the Mean Squared Error (MSE) between actual and predicted values.
  - Parameters:
    - Inputs: Ground truth values (`y_true`) and predicted values (`y_pred`).
    - Optional: Weighting of samples (`sample_weight`).
  - Output: A single float value representing the MSE. Lower values indicate better fit.

- **r2_score(y_true, y_pred)**:
  - Measures the proportion of variance explained by the model. Also known as the coefficient of determination.

  - Parameters:
    - Inputs: Ground truth values (`y_true`) and predicted values (`y_pred`).
    - Optional: Weighting of samples (`sample_weight`).
  - Output: A float value ranging from 0 to 1, with higher values indicating better performance.

- **sm.stats.durbin_watson(residuals)**:
  - Computes the Durbin-Watson statistic to check for autocorrelation in regression residuals.
  - Parameters:
    - Input: Residuals from a regression model (`residuals`).
  - Output: A statistic ranging from 0 to 4. Values near 2 indicate no autocorrelation.

Model study:

Change in *PolynomialFeatures()*:

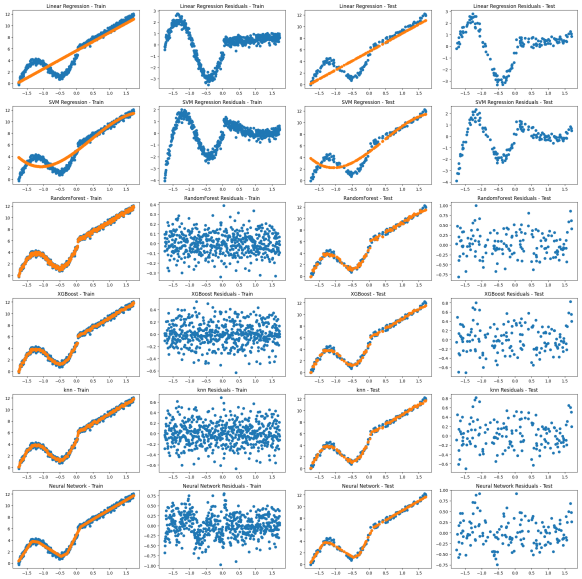

Figure 1: PolynomialFeatures(degree=1)

Train Data

Model	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.833674	1.985063	0.059761	51.466506	6.670987e-12
SVM Regression	0.902665	1.161667	0.102312	50.929773	8.724494e-12

RandomForest	0.999000	0.011934	2.931659	3.769479	1.518686e-01
XGBoost	0.997146	0.034058	2.372918	1.046859	5.924852e-01
knn	0.995917	0.048730	2.529607	0.646017	7.239679e-01
Neural Network	0.993228	0.080821	1.445649	0.445768	8.002078e-01

Test Data

Model	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.818068	2.255404	0.079118	12.500550	0.001930
SVM Regression	0.878717	1.503541	0.119202	4.156199	0.125168
RandomForest	0.991747	0.102311	1.834557	0.904898	0.636068
XGBoost	0.992842	0.088738	1.851431	1.003285	0.605535
knn	0.993137	0.085085	1.982435	1.152556	0.561986
Neural Network	0.991689	0.103035	1.588613	4.565183	0.102019

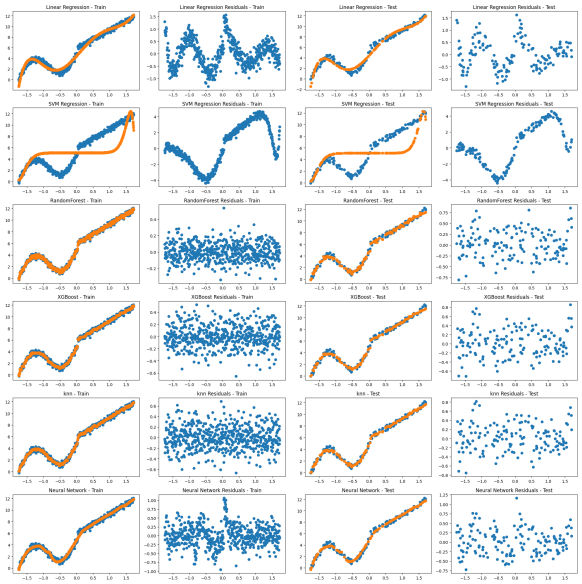

Figure 2: PolynomialFeatures(degree=6)

Train Data

Model	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.976261	0.283320	0.417086	11.050745	3.984384e-03
SVM Regression	0.439366	6.691056	0.018313	33.488925	5.345302e-08
RandomForest	0.998976	0.012218	2.925347	22.983624	1.021338e-05
XGBoost	0.997136	0.034178	2.386600	2.239290	3.263957e-01

knn	0.995876	0.049216	2.513787	0.434186	8.048553e-01
Neural Network	0.992277	0.092176	1.274863	13.848899	9.834445e-04

Test Data

Model	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.972925	0.335644	0.528190	3.916151	0.141130
SVM Regression	0.466641	6.612025	0.030827	8.110347	0.017332
RandomForest	0.991962	0.099646	1.848506	0.436826	0.803794
XGBoost	0.992730	0.090122	1.854642	1.068547	0.586095
knn	0.992941	0.087513	1.992388	0.612068	0.736362
Neural Network	0.991206	0.109019	1.533037	4.171648	0.124205

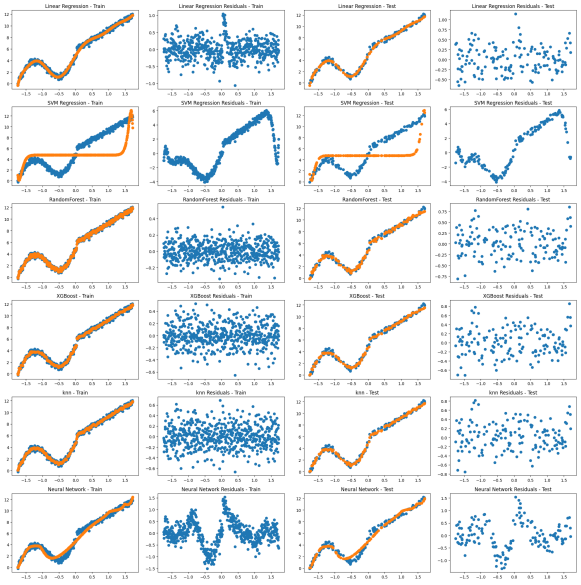

Figure 3: PolynomialFeatures(degree=10)

Train Data

Model	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.833674	1.985063	0.059761	51.466506	6.67e-12
SVM Regression	0.902665	1.161667	0.102312	50.929773	8.72e-12
RandomForest	0.999000	0.011934	2.931659	3.769479	1.52e-01
XGBoost	0.997146	0.034058	2.372918	1.046859	5.92e-01
knn	0.995917	0.048730	2.529607	0.646017	7.24e-01
Neural Network	0.993228	0.080821	1.445649	0.445768	8.00e-01

Test Data

Model	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.818068	2.255404	0.079118	12.500550	0.001930
SVM Regression	0.878717	1.503541	0.119202	4.156199	0.125168
RandomForest	0.991747	0.102311	1.834557	0.904898	0.636068
XGBoost	0.992842	0.088738	1.851431	1.003285	0.605535
knn	0.993137	0.085085	1.982435	1.152556	0.561986
Neural Network	0.991689	0.103035	1.588613	4.565183	0.102019

Comparison of Train MSE for Polynomial Degrees 1 and 10

Model	Train MSE (Degree 1)	Train MSE (Degree 10)	Difference
Linear Regression	1.985063	0.083693	-1.901370
SVM Regression	1.161667	9.000590	+7.838923
RandomForest	0.011934	0.012243	+0.000309
XGBoost	0.034058	0.034178	+0.000120
knn	0.048730	0.049216	+0.000486
Neural Network	0.080821	0.266943	+0.186122

SVM Regression

The training MSE increases dramatically from 1.161667 (degree 1) to 9.000590 (degree 10), a difference of approximately +7.84. This large increase implies that the SVM model—with the selected parameters—is very sensitive to the increased complexity.

Linear Regression

Conversely, Linear Regression shows a significant improvement with the training MSE dropping from 1.985063 (degree 1) to 0.083693 (degree 10) – a decrease of about 1.90. This indicates that when provided with additional polynomial features, the linear model is able to capture the underlying relationship much more effectively.

RandomForest, XGBoost, and k-NN

These methods show negligible changes in training MSE (differences on the order of 0.0001–0.0005). This suggests that their performance is robust to the change in polynomial degree.

Neural Network

The Neural Network's training MSE increases moderately from 0.080821 to 0.266943 (a difference of +0.186122). Although not as dramatic as the SVM, this change still indicates that the network is affected by the higher-degree features, possibly due to increased difficulty in training or slight overfitting.

Standaization:

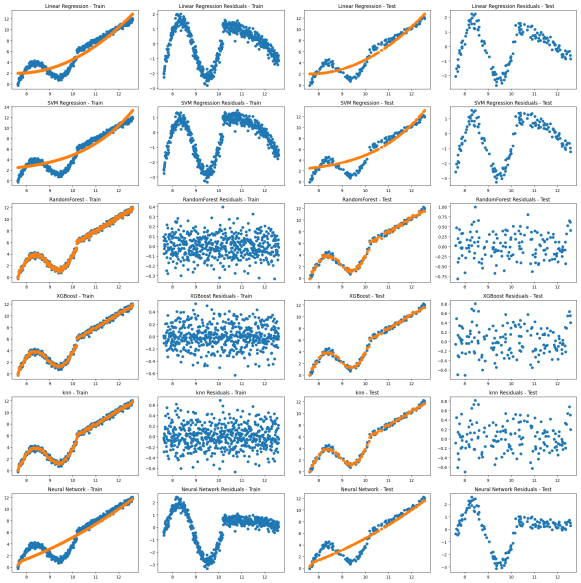

Figure 4: Without standardization)

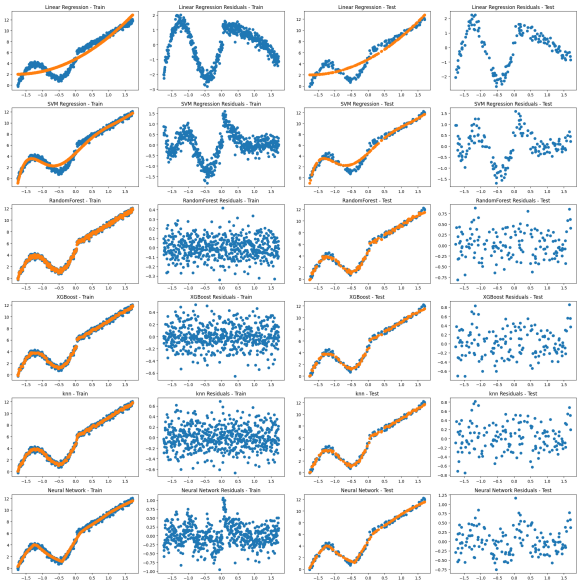

Figure 5: With standardization)

For Train Data:

Model	With Standardization	Without Standardization
Linear Regression	1.333367	1.333367
SVM Regression	0.364652	1.653057
RandomForest	0.011759	0.011919
XGBoost	0.034178	0.034058
knn	0.049216	0.048993
Neural Network	0.093746	1.614388

For Test data:

Model	With Standardization	Without Standardization
Linear Regression	1.483413	1.483413
SVM Regression	0.439551	1.805088
RandomForest	0.099528	0.099423
XGBoost	0.091085	0.086720
knn	0.087513	0.086146
Neural Network	0.115446	1.831578

Observations

- **SVM Regression and Neural Network** are significantly affected by standardization. Their MSE values drop drastically with standardization, indicating that these models perform poorly without feature scaling.
- **Linear Regression** is not affected, as expected, since it does not rely on feature scaling for performance.
- **RandomForest, XGBoost, and k-NN** show negligible changes, indicating they are robust to standardization.

Neural network:

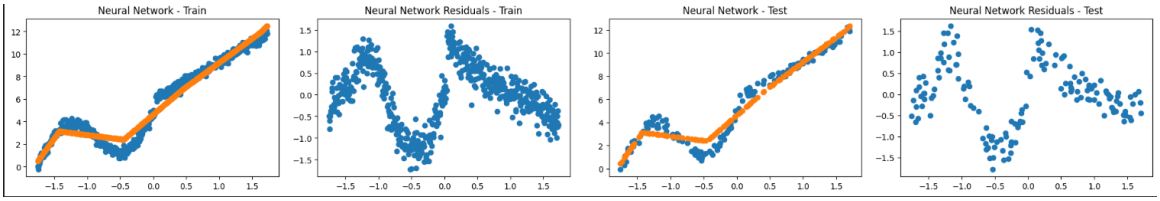

Figure 6: 1 Hidden Layer

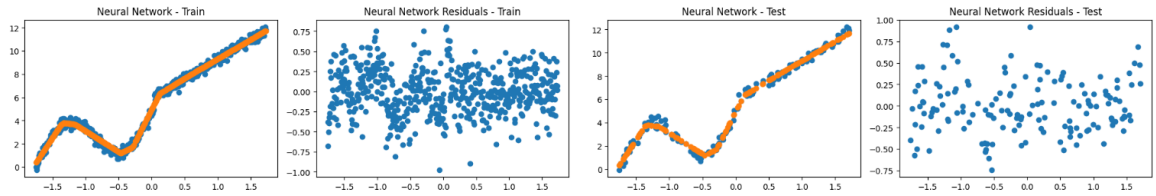

Figure 7: 2 Hidden Layer

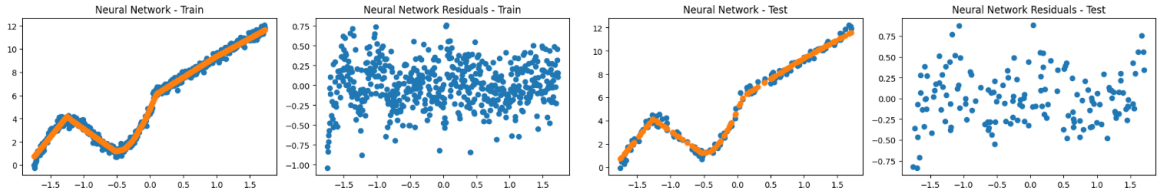

Figure 8: 3 Hidden Layer

Train MSE for Neural Networks with 1, 2, and 3 Hidden Layers

Model	NN1 (1 Layer)	NN2 (2 Layers)	NN3 (3 Layers)
Train MSE	0.494339	0.080821	0.081402

Test MSE for Neural Networks with 1, 2, and 3 Hidden Layers

Model	NN1 (1 Layer)	NN2 (2 Layers)	NN3 (3 Layers)
Test MSE	0.573550	0.103035	0.099799

Observations

- The Neural Network with 1 hidden layer has the highest MSE values for both train and test data, indicating underfitting.
- Increasing the number of layers significantly reduces MSE, improving the model's ability to capture patterns in the data.
- The difference in MSE between 2 and 3 hidden layers is minimal, suggesting diminishing returns when adding more layers.
- The best performance is observed with 2 or 3 hidden layers, as they balance complexity and generalization.

Outliers:

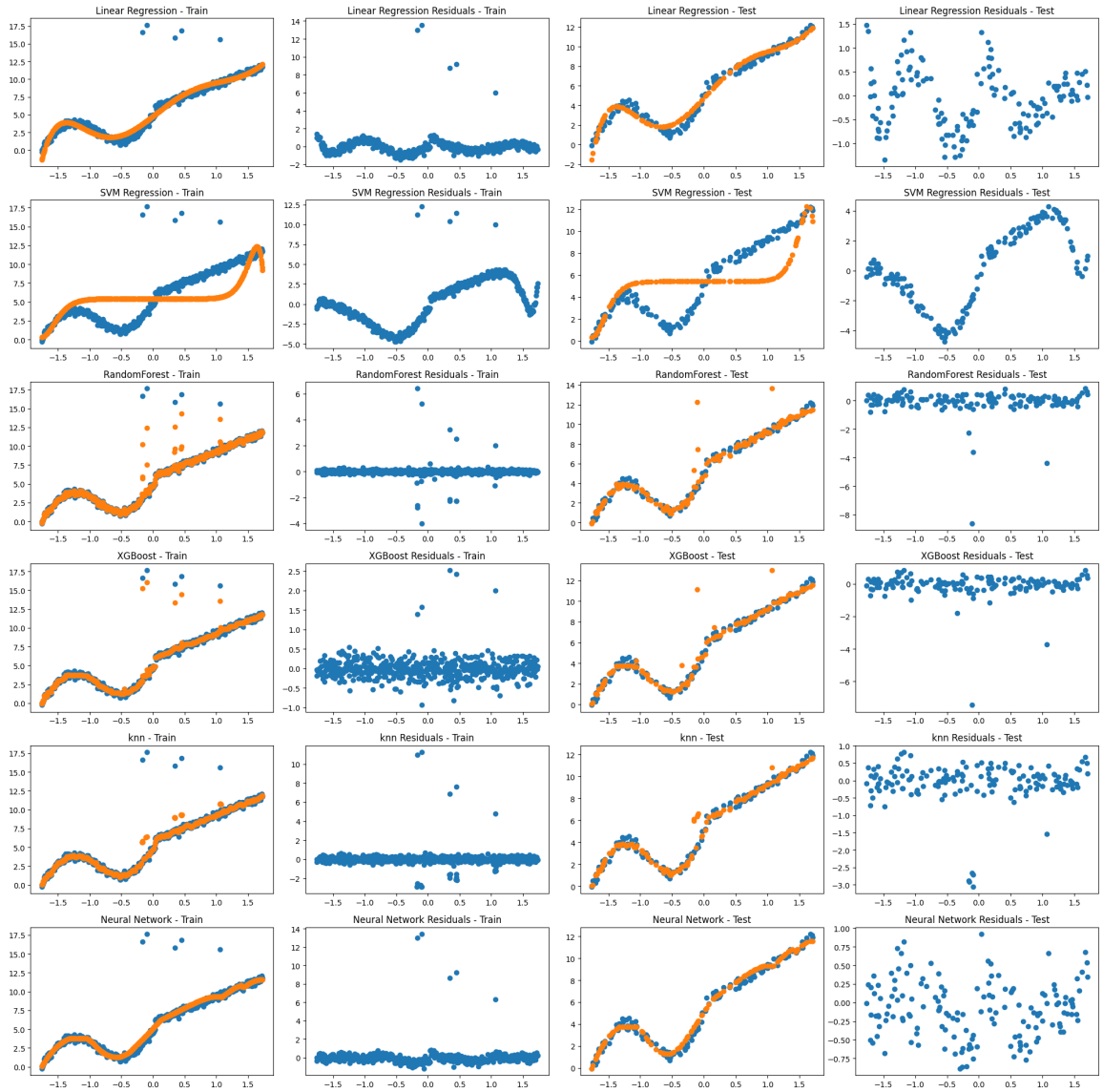

Figure 9: Metrics when PolynomialFeatures(degree=10)

Observation

- **Linear Regression** doesn't captures the presence of outliers
- **SVR** too doesn't capture outliers
- **Random Forest** Does capture presence of outleirs to a very good extent

- **XGBoost** is similar to the above
- **KNN** Does capture them but now to a good extent
- **Neural Networks** is able to ignore the presence of outliers

From this is we know the data is noisy and don't wish to cover it then the models like linear regression and Neural Networks wold be good.

Limitations on non parametric methods:

- **Dimensionality:** Non-parametric methods like k-NN can perform poorly as the number of features grows. Distances in high-dimensional spaces become less meaningful, leading to degraded performance.
- **Computational Complexity:** Methods such as k-NN require storing the entire dataset and computing distances to all training samples during prediction, making them computationally expensive for large datasets.
- **Overfitting Tendencies:** Decision trees can easily overfit if not properly pruned or regularized, capturing noise in the data.
- **Sensitivity to Noise and Outliers:** Non-parametric models can be heavily influenced by outliers in the local neighborhood (e.g., k-NN). This can distort distance calculations and lead to inaccurate predictions.
- **Scalability and Memory:** Storing large datasets for k-NN can be impractical, and querying can become prohibitively slow.

Should linear regression be used?

Linear Regression can still be a good choice in several scenarios:

- **Data is (Largely) Linear:** If exploratory analysis suggests that relationships between predictors and the target are close to linear, then Linear Regression is both simple and effective.
- **Low Variance, High Bias Tolerance:** Linear Regression has relatively low variance. If you have a smaller dataset or prefer a more stable model that is less prone to overfitting, Linear Regression is advantageous.
- **Interpretability and Speed:** Coefficients in a linear model are easy to interpret, and fitting is computationally efficient compared to more complex models like XGBoost or Neural Networks.

Justification: Even though advanced models (e.g., tree-based methods, SVMs, Neural Networks) often achieve better performance on complex data, Linear Regression remains relevant when:

1. The underlying relationship is (approximately) linear.
2. The dataset is not excessively large or high-dimensional.
3. Interpretability is crucial for the application domain.

Review of SVR and Random forest:

SVR:

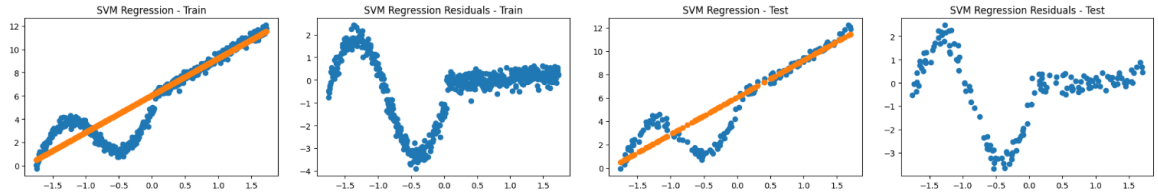

Figure 10: When Kernel = linear

Metrics - Train Data:

	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.833674	1.985063	0.059761	51.466506	6.670987e-12
SVM Regression	0.822859	2.114138	0.056112	47.489020	4.874055e-11
RandomForest	0.999000	0.011934	2.931659	3.769479	1.518686e-01
XGBoost	0.997146	0.034058	2.372918	1.046859	5.924852e-01
knn	0.995917	0.048730	2.529607	0.646017	7.239679e-01
Neural Network	0.993179	0.081402	1.440899	6.018382	4.933158e-02

Metrics - Test Data:

	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.818068	2.255404	0.079118	12.500550	0.001930
SVM Regression	0.809247	2.364748	0.075482	11.580112	0.003058
RandomForest	0.991747	0.102311	1.834557	0.904898	0.636068
XGBoost	0.992842	0.088738	1.851431	1.003285	0.605535
knn	0.993137	0.085085	1.982435	1.152556	0.561986
Neural Network	0.991950	0.099799	1.670016	1.572797	0.455482

Figure 11: When Kernel = linear

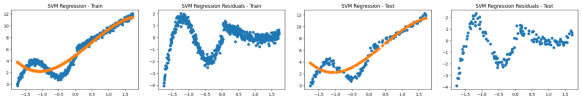

Figure 12: When Kernel = poly

Metrics - Train Data:

	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.833674	1.985063	0.059761	51.466506	6.670987e-12
SVM Regression	0.902665	1.161667	0.102312	50.929773	8.724494e-12
RandomForest	0.999000	0.011934	2.931659	3.769479	1.518686e-01
XGBoost	0.997146	0.034058	2.372918	1.046859	5.924852e-01
knn	0.995917	0.048730	2.529607	0.646017	7.239679e-01
Neural Network	0.993179	0.081402	1.440899	6.018382	4.933158e-02

Metrics - Test Data:

	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.818068	2.255404	0.079118	12.500550	0.001930
SVM Regression	0.878717	1.503541	0.119202	4.156199	0.125168
RandomForest	0.991747	0.102311	1.834557	0.904898	0.636068
XGBoost	0.992842	0.088738	1.851431	1.003285	0.605535
knn	0.993137	0.085085	1.982435	1.152556	0.561986
Neural Network	0.991950	0.099799	1.670016	1.572797	0.455482

Figure 13: When Kernel = poly

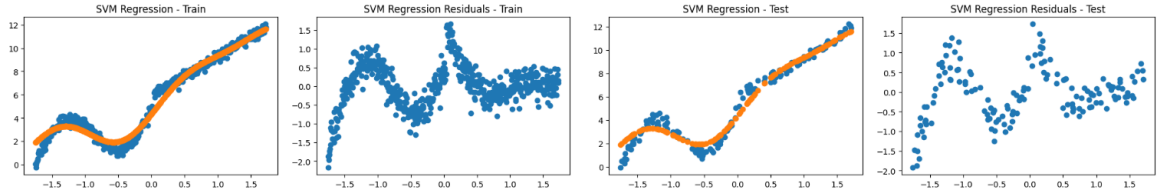

Figure 14: When Kernel = rbf

Metrics - Train Data:

	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.833674	1.985063	0.059761	51.466506	6.670987e-12
SVM Regression	0.972519	0.327981	0.358821	28.038267	8.157701e-07
RandomForest	0.999000	0.011934	2.931659	3.769479	1.518686e-01
XGBoost	0.997146	0.034058	2.372918	1.046859	5.924852e-01
knn	0.995917	0.048730	2.529607	0.646017	7.239679e-01
Neural Network	0.993179	0.081402	1.440899	6.018382	4.933158e-02

Metrics - Test Data:

	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.818068	2.255404	0.079118	12.500550	0.001930
SVM Regression	0.964298	0.442592	0.376628	0.673661	0.714030
RandomForest	0.991747	0.102311	1.834557	0.904898	0.636068
XGBoost	0.992842	0.088738	1.851431	1.003285	0.605535
knn	0.993137	0.085085	1.982435	1.152556	0.561986
Neural Network	0.991950	0.099799	1.670016	1.572797	0.455482

Figure 15: When Kernel = rbf

Randomforest:

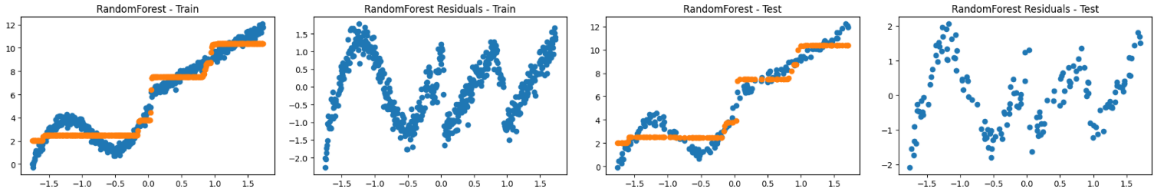

Figure 16: When max depth = 2

Metrics - Train Data:

	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.833674	1.985063	0.059761	51.466506	6.670987e-12
SVM Regression	0.902665	1.161667	0.102312	50.929773	8.724494e-12
RandomForest	0.939277	0.724721	0.162514	20.325373	3.858348e-05
XGBoost	0.997146	0.034058	2.372918	1.046859	5.924852e-01
knn	0.995917	0.048730	2.529607	0.646017	7.239679e-01
Neural Network	0.993179	0.081402	1.440899	6.018382	4.933158e-02

Metrics - Test Data:

	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.818068	2.255404	0.079118	12.500550	0.001930
SVM Regression	0.878717	1.503541	0.119202	4.156199	0.125168
RandomForest	0.931370	0.850801	0.247164	3.548577	0.169604
XGBoost	0.992842	0.088738	1.851431	1.003285	0.605535
knn	0.993137	0.085085	1.982435	1.152556	0.561986
Neural Network	0.991950	0.099799	1.670016	1.572797	0.455482

Figure 17: When ma depth = 2

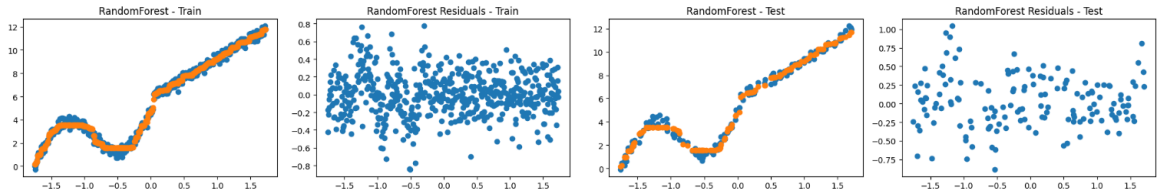

Figure 18: When max depth = 5

Metrics - Train Data:

	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.833674	1.985063	0.059761	51.466506	6.670987e-12
SVM Regression	0.902665	1.161667	0.102312	50.929773	8.724494e-12
RandomForest	0.994864	0.061302	1.649352	6.683040	3.538314e-02
XGBoost	0.997146	0.034058	2.372918	1.046859	5.924852e-01
knn	0.995917	0.048730	2.529607	0.646017	7.239679e-01
Neural Network	0.993179	0.081402	1.440899	6.018382	4.933158e-02

Metrics - Test Data:

	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.818068	2.255404	0.079118	12.500550	0.001930
SVM Regression	0.878717	1.503541	0.119202	4.156199	0.125168
RandomForest	0.990406	0.118934	1.415535	1.557616	0.458953
XGBoost	0.992842	0.088738	1.851431	1.003285	0.605535
knn	0.993137	0.085085	1.982435	1.152556	0.561986
Neural Network	0.991950	0.099799	1.670016	1.572797	0.455482

Figure 19: When max depth = 5

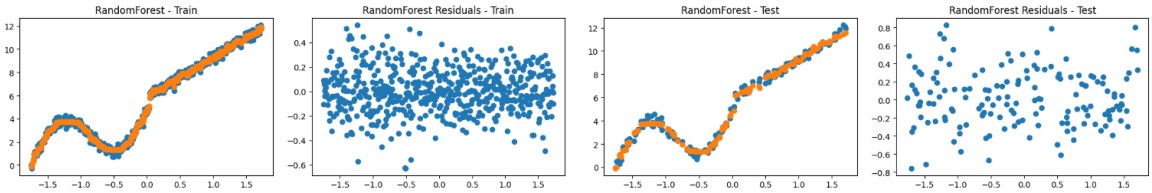

Figure 20: When max depth = 7

Metrics - Train Data:

	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.833674	1.985063	0.059761	51.466506	6.670987e-12
SVM Regression	0.902665	1.161667	0.102312	50.929773	8.724494e-12
RandomForest	0.997254	0.032769	2.403271	5.391926	6.747736e-02
XGBoost	0.997146	0.034058	2.372918	1.046859	5.924852e-01
knn	0.995917	0.048730	2.529607	0.646017	7.239679e-01
Neural Network	0.993179	0.081402	1.440899	6.018382	4.933158e-02

Metrics - Test Data:

	R-squared	MSE	Durbin-Watson	Jarque-Bera	JB P-value
Linear Regression	0.818068	2.255404	0.079118	12.500550	0.001930
SVM Regression	0.878717	1.503541	0.119202	4.156199	0.125168
RandomForest	0.992284	0.095656	1.804407	0.875271	0.645561
XGBoost	0.992842	0.088738	1.851431	1.003285	0.605535
knn	0.993137	0.085085	1.982435	1.152556	0.561986
Neural Network	0.991950	0.099799	1.670016	1.572797	0.455482

Figure 21: When max depth = 7

Lasso and ridge regression:

Ridge()

- Linear least squares with L2 regularization.
- Helps prevent overfitting by adding a penalty term.

Lasso()

- Linear Model trained with L1 prior as regularizer (Lasso).
- Useful for feature selection as it can shrink some coefficients to zero.

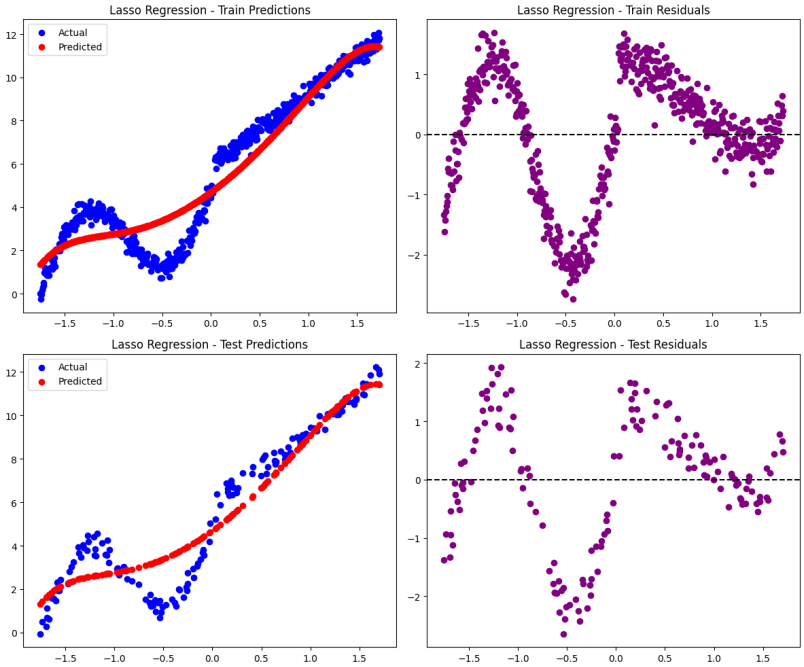

Figure 22: Lasso Regression output

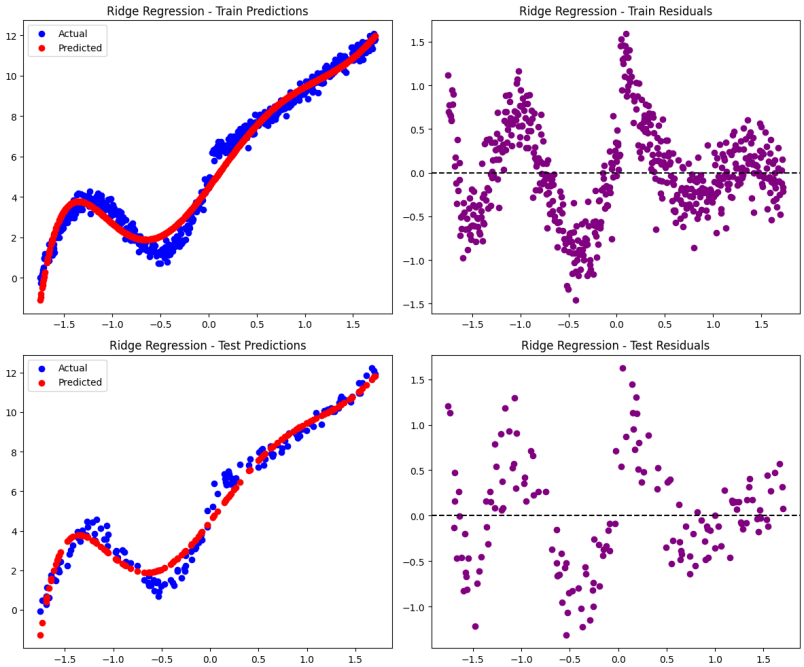

Figure 23: Ridge Regression output

Metrics for Ridge Regression

Metric	Train Value	Test Value
R-squared	0.9754915466	0.9715399673
MSE	0.2925033363	0.3528172869
Durbin-Watson	0.4038265642	0.4987455420
Jarque-Bera	3.8007391672	2.6031091847
JB P-value	0.1495133513	0.2721084463

Metrics for Lasso Regression

Metric	Train Value	Test Value
R-squared	0.9095462433	0.9007501803
MSE	1.0795469293	1.2303939541
Durbin-Watson	0.1097164771	0.1426785244
Jarque-Bera	42.5901457088	7.4572095288
JB P-value	5.6450444712e-10	0.0240263349