Laptop Price Predictor

EE 559 Course Project
Data Set:Laptop Prices

Tanvi Gandhi, Sumukh Shetty, tanvibhu@usc.edu, sumukhsh@usc.edu

April 27, 2024

1 Abstract

Laptops have become indispensable tools for a variety of daily tasks, leading to a saturated market where manufacturers and consumers alike face challenges. As manufacturers struggle to differentiate their products, consumers find it increasingly difficult to make informed choices. This paper introduces a novel machine learning (ML) approach to address these challenges by enhancing the accuracy of laptop price predictions. We employ a range of ML models—including Regularized Linear Regression, Bayesian methods, RBF Network, and Polynomial Regression—analyzing their efficacy with diverse feature combinations. Our study not only tests the performance of each model but also refines the feature identification process to improve prediction precision. This research aims to demonstrate how advanced ML techniques can be strategically applied to predict laptop prices more accurately, offering insights into both methodological advancements and practical applications in the evolving landscape of technology sales.

2 Introduction

2.1 Problem Assessment and Goals

The Laptop Prices dataset has been selected for this project, a regression problem that aims to predict laptop prices based on their specifications. The dataset contains 1300 data points and 9 features, including screen size, CPU, RAM, storage, graphics card, operating system, weight, and price. This dataset provides a comprehensive resource for understanding how laptop specifications affect prices across various brands and models.

The goals of this project are:

- To develop a model that can accurately predict laptop prices based on their specifications
- To analyze how different features (such as screen size, CPU, RAM, etc.) affect laptop prices
- To compare the performance of different reference systems (trivial, 1NN, and Linear Regression) on this dataset
- To achieve a low Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and a high R-squared (R2) value
- To apply data preprocessing and feature engineering techniques to improve model performance

2.2 Literature Review

3 Approach and Implementation

Report your approach and implementation details in the following subsections. You should mention which libraries you used, and which functions you coded yourself in section 5, but avoid including

code anywhere else in your report. Your description of what your system does should be readable and understandable to a reader that isn't familiar with the functions and libraries you used but is familiar with the algorithms and techniques that were covered in EE 559. (For example, stating "we standardized all real-valued features", and also stating the functions used in your code for this, is fine; stating only the functions used in your code is not fine.)

3.1 Dataset Usage

3.1.1 Data Partitioning

- **Total Data Points:** The dataset consisted of 1300 data points.
- **Training Data:** 900 data points were allocated for training.
- **Testing Data:** The remaining 400 data points were reserved exclusively for testing to evaluate the model's performance on unseen data.

3.1.2 Preprocessing

The preprocessing phase was essential in preparing the dataset for model training and evaluation:

- **Feature Engineering:** The entire training dataset was analyzed to apply one-hot encoding to categorical variables based on the unique values in specific columns. This step was critical for converting categorical text data into a numerical format usable by machine learning algorithms.
- **Handling Inconsistencies in Test Data:** To ensure consistency between the training and test datasets following one-hot encoding:
  - *Adding Missing Columns:* Columns found in the training dataset but absent in the test dataset were added to the latter, with all values set to zero.
  - *Removing Extra Columns:* Columns present in the test dataset but not in the training dataset were removed.

3.1.3 Training and Validation Strategy

- **Bayesian Regression:** Utilized the full training set of 900 data points without partitioning into subsets.
- **Ridge Regression and RBF Networks:**
  - **Validation Sets:** The training data was split further to include a validation set constituting 20% (180 data points), used for fine-tuning model parameters.
  - **Cross-Validation:**
    - *K-Fold Setup:* Employed a 5-fold cross-validation, dividing the training data, including the validation segment, into five equal parts.
    - *Nested Loops:* In RBF models, sequential loops were used to optimize the gamma value and find the best (K, gamma) pair, ensuring each loop focused on separate parameter adjustments based on validation results.
    - *Standardization:* Performed inside the cross-validation loop to avoid data leakage and ensure the integrity of parameter scaling.
  - **Parameter Optimization:** The 5-fold cross-validation was pivotal in determining suitable regularization strengths (alpha) for Ridge regression, subsequently applied to ensure optimal model regularization.

3.1.4 Testing Strategy

- **Utilization of Test Data:** All 400 data points in the test dataset were employed to evaluate the final models, providing a thorough assessment of performance against new, unseen data.

3.1.5 Decision Making Based on Validation Sets

- **Role of Validation in Model Tuning:** Validation sets were crucial in tuning model parameters, with decisions regarding the optimal settings for parameters like gamma in RBF Networks and alpha in Ridge regression being made based on computed performance metrics (RMSE, MAE, or R^2).
- **Feedback from Cross-Validation:** Results from cross-validation directly influenced adjustments in model parameters, ensuring models were well-fitted to the training data and generalized effectively to new data.

3.2 Preprocessing and Exploratory Data Analysis

Log-Normal Transformation on Price Column: The distribution of the target variable, price, was observed to be right-skewed. To improve the performance of the algorithm, a log-normal transformation is applied, which effectively normalizes the distribution. This transformation involves taking the logarithm of the price values, as shown below. Therefore, when separating dependent and independent variables, we apply a logarithmic transformation to the price. To display the results in their original scale, we subsequently take the exponent of the predicted values.

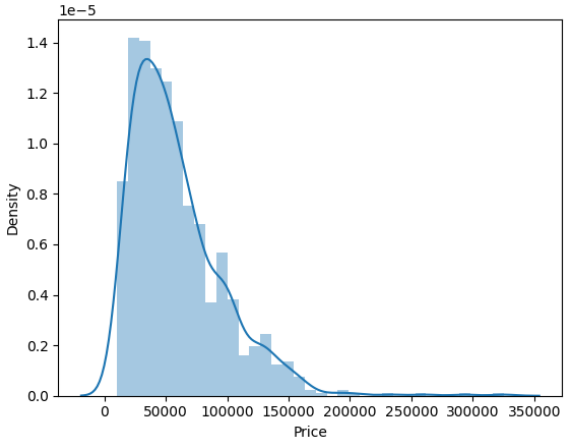

Figure 1: Right skewed

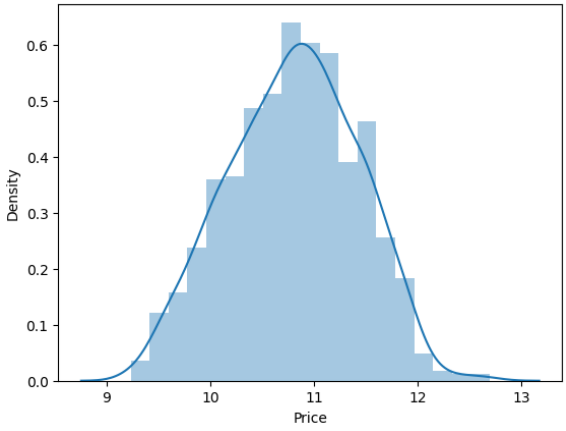

Figure 2: Normal Distribution

Number column dropped: The 'Number' column, which primarily consisted of index values, was dropped from the dataset. This decision was made because it had very little correlation with the target variable, price, and no relevant relation to the outcomes being predicted. Dropping this column helps streamline the model by removing unnecessary features.

Inches Column :

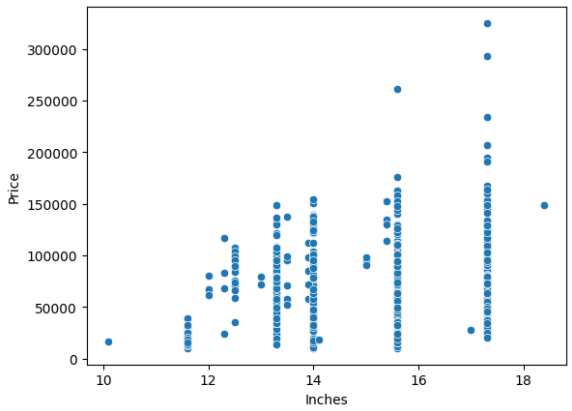

Figure 3: Price Variation with laptop size in Inches

From the plot we can conclude that there is a relationship but not a strong relationship between the price and size column

Company:

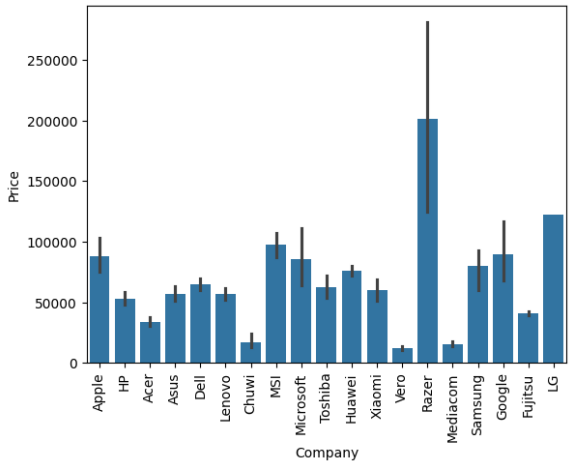

Figure 4: Price variation as per Brands

Screen Resolution: Observed Display types: 'IPS Panel', 'Full HD', 'Touchscreen', '4K Ultra HD', 'Quad HD+', 'Retina Display', If we plot the "Touch Screen" column against price, we find that laptops with touch screens tend to be more expensive, which reflects real-life pricing trends.

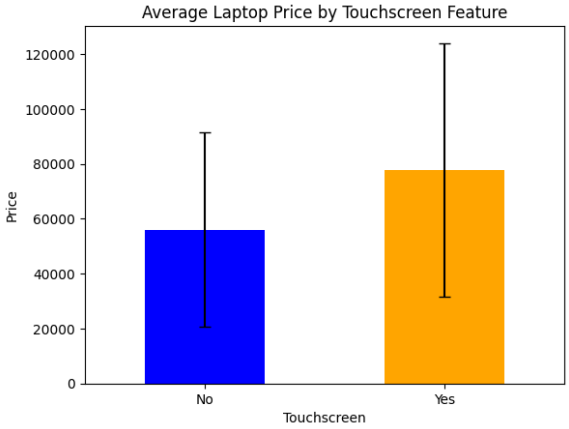

Figure 5: Price Variation if TouchScreen or not

CPU:

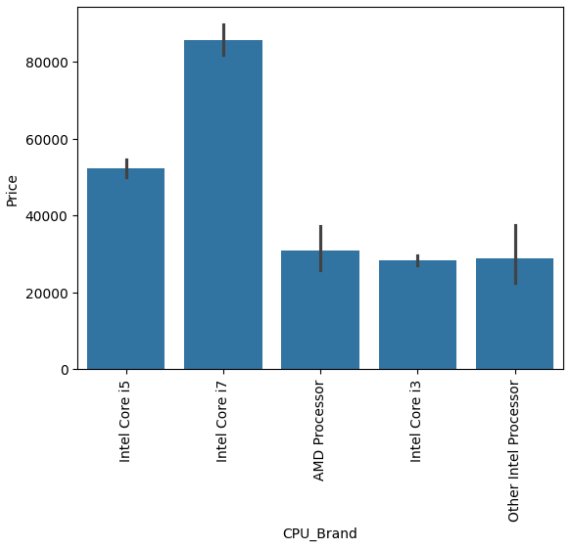

Figure 6: CPU Company Price Variation

GPU:

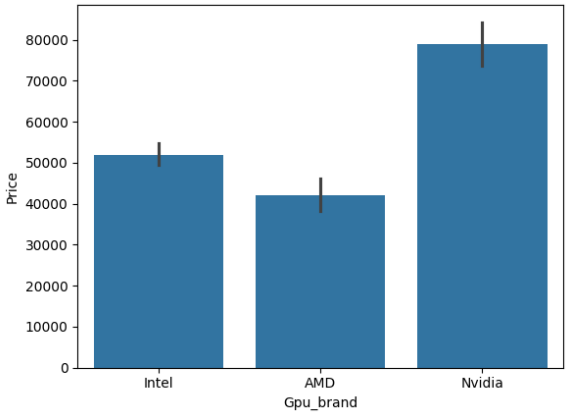

Figure 7: GPU brand Price Variation

On the basis on this, Feature Engineering was done.

1.

3.3 Feature Engineering

The preprocessing steps applied to the dataset are as follows:

1. **One-Hot Encoding of 'Opsys' Column:** This transformation created 7 additional features, each representing a distinct operating system category. One-hot encoding was chosen to convert categorical data into a numerical format suitable for machine learning algorithms.

2. **One-Hot Encoding of 'Company Name' Column:** This process expanded the column into 17 additional features, converting nominal company names into numerical values for compatibility with machine learning algorithms.

3. **One-Hot Encoding of 'TypeName' Column:** The column representing laptop types was one-hot encoded into 6 additional features, facilitating numerical representation of categorical data for machine learning models.

4. **Division of 'Screen Resolution' Column:** This step split the 'Screen Resolution' column into two separate columns, 'displayName' and 'Resolution', for better organization and clarity of information. Subsequently, the 'Resolution' column was split into 'Width' and 'Height' columns, and the 'Resolution' column was dropped from the DataFrame. Additionally, the 'PPI' (Pixels Per Inch) feature was computed based on the width, height, and screen size (Inches) to provide a more granular representation of screen quality.

5. **Splitting of 'CPU' Column:** The 'CPU' column was divided into two separate columns: one containing the CPU name and the other containing the CPU clock speed, aiding in the extraction of relevant information for analysis.

6. **Processing of 'CPU Name' Column:** The 'CPU name' column was refined to categorize CPU models into broader categories, such as 'Intel Core i5', 'Intel Core i3', 'Intel Core i7', 'AMD Processor', and 'Other Intel Processor', to reduce feature dimensionality while preserving essential information. This was followed by one-hot encoding of the processed 'CPU Name' column.

7. **Splitting of 'Memory' Column:** The 'Memory' column was split into three separate columns, namely 'SSD', 'HDD', and 'Flash Storage', each containing the respective memory capacity of the specified type. This division facilitated more granular analysis of memory specifications within the dataset.

8. **Extraction of Features from 'displayName' Column:** Features such as 'IPS Panel', 'Full HD', 'Touchscreen', '4K Ultra HD', 'Quad HD+', and 'Retina Display' were extracted from the 'displayName' column using a binary encoding approach. Each feature was then converted into a separate column, with a value of 1 indicating the presence of the feature and 0 indicating its absence. The original 'displayName' column was subsequently dropped from the DataFrame.

9. **Extraction of Features from 'Gpu' Column:** GPU brand information was extracted from the 'Gpu' column, and a new 'Gpu_brand' column was created to represent this data. One-hot encoding was then applied to the 'Gpu_brand' column to convert categorical GPU brand data into numerical format for machine learning algorithms.

10. **Conversion of 'CPU_Clock_Speed' Column:** The 'CPU_Clock_Speed' column, originally represented as a string, was converted to float type to facilitate numerical operations and analysis.

11. **Created Pixels per Inches Column:** Features such as Height, width were extracted from the 'Resolution' column. Each feature's (Height, Width, inches) correlation was then checked against the 'Price' column and it happened to be very less. A new column was made to include the pixels per inches which had a better correlation with the Price column. The original 'Inches' column and the newly created Height, Width columns were then dropped from the DataFrame.

These preprocessing techniques were chosen to transform and structure the data in a manner conducive to effective analysis and modeling, ensuring compatibility with various machine learning algorithms while preserving relevant information from the original dataset.

If you did not develop any new features, state "Not applicable" for this section.

3.4 Feature Dimensionality Adjustment

To address potential issues related to high-dimensional feature spaces and multicollinearity, Principal Component Analysis (PCA) was employed. PCA is a dimensionality reduction technique that projects high-dimensional data onto a lower-dimensional subspace while retaining the maximum variance. This aids in simplifying the dataset and improving model performance.

The following steps were implemented for PCA:

- **Standardization of Data:** Prior to applying PCA, the data was standardized using the StandardScaler from scikit-learn. Standardization ensures that all features have a mean of 0 and a standard deviation of 1, preventing features with larger scales from dominating the analysis.

- **PCA Application:** PCA was then applied with the objective of retaining 95% of the variance in the dataset. This choice of variance retention threshold strikes a balance between reducing dimensionality and preserving the majority of the original information.

- **Number of Components Determination:** The number of principal components required to retain 95% of the variance in the dataset was automatically determined during the PCA fitting process.

- **Transformation of Data:** The original data was transformed into the lower-dimensional space defined by the principal components. The transformed data, represented by the principal components, was then used for subsequent modeling tasks.

The PCA implementation was integrated into the preprocessing pipeline to reduce the dimensionality of the feature space while preserving the majority of the information. This facilitates improved model training and inference efficiency without significant loss of predictive power.

Model	Before PCA	After PCA
Linear Regression with Regularization	RMSE: 0.277 MAE: R2: 0.806	RMSE: 0.301 MAE: 0.230 R2: 0.769
SVR Algorithm - RBF Kernel + Linear Regression (regularized)	RMSE: 0.244 MAE: 0.189 R2: 0.850	RMSE: 0.299 MAE: 0.229 R2: 0.773
Bayesian Multivariate Linear Regression	RMSE: 0.449 MAE: 0.281 R2: 0.488	RMSE: 0.303 MAE: 0.230 R2: 0.767
Polynomial Regression	RMSE: 1.154 MAE: 0.521 R2: -2.379	RMSE: 378083077227.695 MAE: 96572071247.084 R2: -362708758945519626092544.000
Polynomial Regression (with Regularization)	RMSE: 8800.615 MAE: 5811.472 R2: -196521271.507	RMSE: 7.147 MAE: 3.954 R2: -128.615

Table 1: Performance Results of models before and after Dimensionality reduction

3.5 Training, Classification or Regression, and Model Selection

3.5.1 Linear Regression with Regularization

Implemented a Ridge regression model to determine the optimal regularization parameter (alpha) that minimizes the root mean square error (RMSE) on a validation dataset. It iterates through a predefined list of alpha values, fits the Ridge model for each alpha using training data , and evaluates each model's performance on a validation set. After determining the alpha that results in the lowest RMSE, it retrains the model using this optimal alpha on the training data and evaluates its performance on a separate test dataset. The code prints out the best alpha value, the lowest RMSE on the validation data, and the RMSE on the test data, and plots to compare actual vs predicted value.

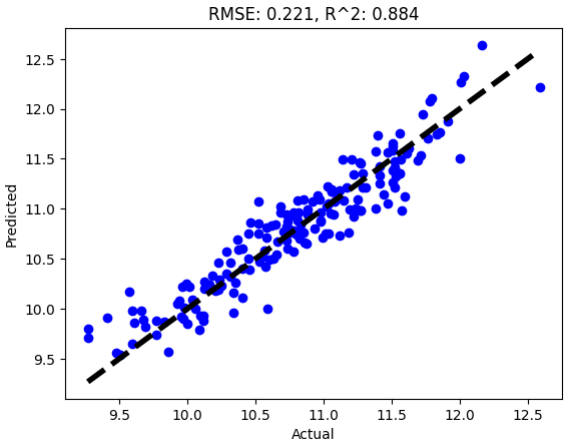

Figure 8: Linear Regression with Regularization: Performance on Test Data

3.5.2 SVR Algorithm - Linear Kernel

For training our Support Vector Regression (SVR) model, we opted for a linear kernel due to its simplicity and effectiveness for our dataset. The main parameter we focused on tuning was the regularization parameter C, which controls the trade-off between minimizing the training error and maximizing the margin. To determine the optimal C value, we employed a grid search approach using the GridSearchCV function from the scikit-learn library. We constructed a parameter grid consisting of various C values ranging from 0.1 to 10,000 and performed 5-fold cross-validation during the grid search process. This allowed us to evaluate the model's performance across different C values and select the one that yielded the lowest mean squared error (MSE) on the validation sets. After completing the grid search, we obtained the best estimator with the optimal C value. We then trained this model on the entire training set and assessed its performance on the test set. The mean squared error (MSE) was calculated to quantify the model's predictive accuracy. Additionally, to provide a visual representation of the relationship between the C values and the MSE, we plotted a graph using matplotlib. This plot helped us visualize how changes in C affected the model's performance and facilitated the selection of an appropriate regularization parameter. From our experiments, we found that the selected C value resulted in a satisfactory MSE on the test set, indicating that our SVR model with a linear kernel performed adequately for our task.

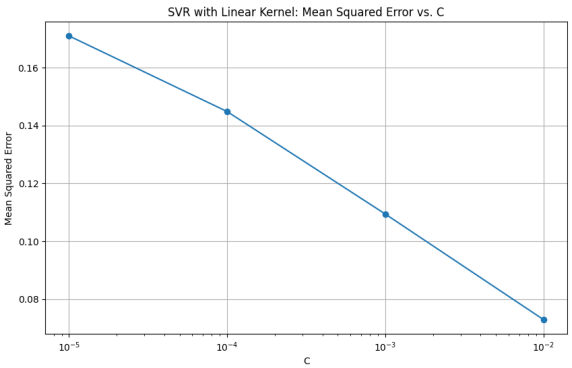

Figure 9: SVR Linear Kernel: Optimizing C Valuesn

3.5.3 SVR Algorithm - RBF Kernel + Linear Regression (regularized)

In the project, we implemented a Ridge Regression model enhanced with Radial Basis Function (RBF) kernel transformations to predict outcomes based on the given dataset. The process involved several key steps to optimize and evaluate the model's performance. Initially, the data was standardized using 'StandardScaler' to ensure uniform scaling. Subsequently, we employed RBF transformations on the scaled data, testing various 'gamma' values to determine the optimal transformation that minimizes validation errors. This transformation allows the linear model to capture non-linear relationships in the data.

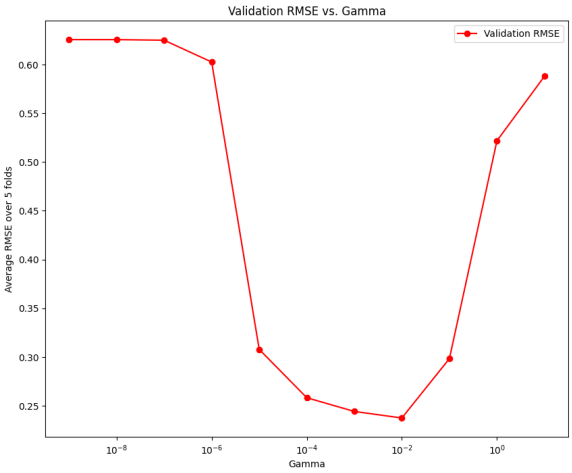

Figure 10: Validation RMSE vs Gamma

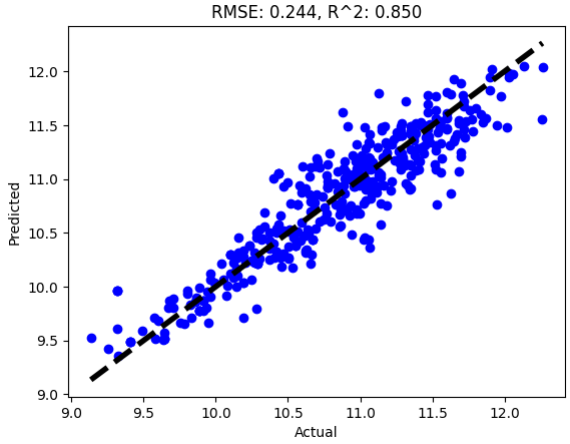

Figure 11: Validation RMSE vs Gamma

Ridge Regression, known for its regularization capabilities, was applied to the transformed data to prevent overfitting, controlled by the regularization parameter 'alpha'. The model's performance was rigorously evaluated using K-Fold cross-validation (5 folds), which helps in assessing the model's effectiveness and stability across different data subsets. The RMSE (Root Mean Square Error) was calculated for each combination of 'gamma', and the best performing model was identified based on the lowest validation RMSE. The selected model was then used to make predictions on new, unseen data, ensuring that the final model is both accurate and generalizable. This comprehensive approach, combining RBF transformations with regularized linear regression within a cross-validated framework, underpins our methodology for developing robust predictive models.

3.5.4 SVR Algorithm - RBF Kernel using Kmeans + Linear Regression

The function optimizes an RBF kernel-based regression model by employing K-means clustering combined with linear regression. The primary objective of this function is to determine the optimal number of cluster centers (K) and the gamma parameter for the RBF kernel, aiming to minimize the root mean square error (RMSE) on validation sets. This tuning enhances the model's accuracy. For each K and gamma combination, K-means clustering is executed

multiple times to identify the best cluster centers, judged by the RMSE from predictions made by a linear regression model trained on RBF-transformed data. A 5-fold cross-validation process manages the training and validation data splits, ensuring that the model's performance is robust and not overly fitted to a specific data subset. This approach is adept at predicting outcomes in datasets with complex, nonlinear relationships by exploiting the RBF kernel's flexibility to effectively model such intricacies. Importantly, this method also helps in reducing the number of RBF centers. By optimizing the number of cluster centers, the approach decreases the complexity and computational demands of the model. Fewer RBF centers mean fewer neurons in the model, leading to less computational overhead and potentially faster prediction times, making it an efficient strategy for building predictive models under non-linearity assumptions in feature-to-target relationships.

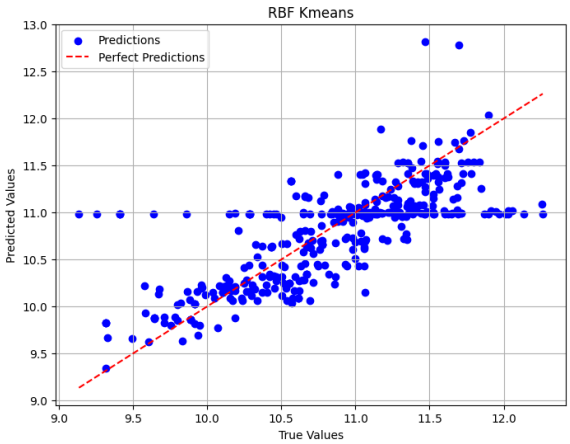

Figure 12: SVR Algorithm - RBF Kernel using Kmeans + Linear Regression: Predicted vs Actual Values on Test Set

3.5.5 Bayesian Multivariate Linear Regression

We implemented Bayesian linear regression to merge prior knowledge with observed data for informed estimates of regression coefficients and predictions. This method provides estimates and adjusts their certainty based on prior beliefs and data evidence. Priors are set for the model parameters: the intercept (assumed to have a normal distribution with mean 0 and standard deviation 10) and slope coefficients (zero-centered normal priors with a standard deviation of 10), indicating substantial uncertainty. The likelihood of the model parameters is initially estimated using normal equations for the beta coefficients. Bayesian updates are then applied, combining these estimates with prior distributions and observed variance of the target variable to refine the posterior means and covariance of the coefficients. Predictions utilize these Bayesian estimates, incorporating parameter uncertainty. While effective, this method faces computational limits with large datasets due to matrix inversion needs. In such cases, platforms like TensorFlow Probability could offer scalable solutions, though not applied here due to dependencies.

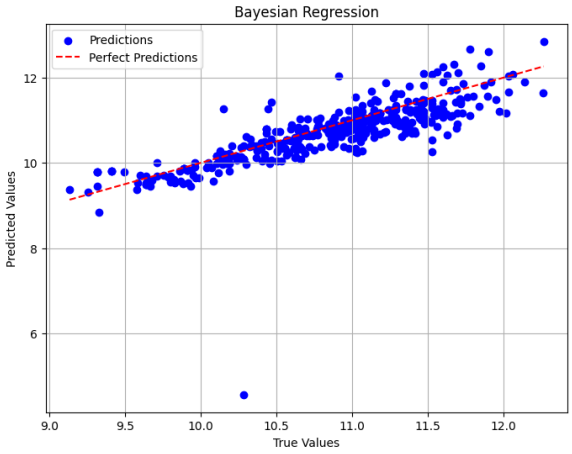

Figure 13: Bayesian Multivariate Linear Regression Performance: Actual vs Predicted Values

3.5.6 PCA Dimensionality Reduction and Bayesian Linear Regression

We implemented a pipeline for dimensionality reduction using Principal Component Analysis (PCA), a fundamental technique in machine learning for simplifying datasets while retaining essential information. PCA was applied to the training data to reduce its dimensionality, successfully preserving 97The PCA model, fitted on the training data, was subsequently applied to the test data to ensure consistency in dimensionality across both datasets. This alignment is essential for accurate model evaluation and performance. The dimensionally reduced dataset was then utilized across various predictive models, including Linear Regression with and without regularization, Support Vector Regression (SVR), and Bayesian Regression, enhancing their performance by focusing on the most informative features.

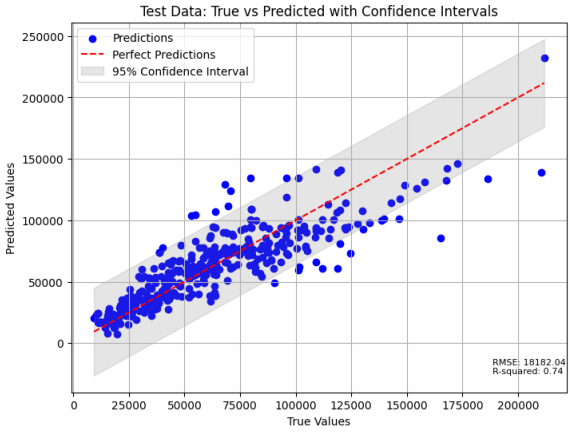

Figure 14: Bayesian Multivariate Linear Regression Performance after PCA: Actual vs Predicted Values

3.5.7 Polynomial Regression

. In this analysis, we implemented a polynomial regression model to investigate the relationship between our variables. Polynomial regression is a form of regression analysis in which the relationship between the independent variable and the dependent variable is modeled as an nth degree polynomial. It extends linear regression by introducing terms with powers greater than one of the predictor variables, allowing the model to capture non-linear relationships more effectively. This method is particularly useful in scenarios where the data exhibits curvature that a simple linear model cannot adequately fit. Polynomial features up to the second degree are generated for both training and testing datasets using Scikit-learn's PolynomialFeatures. This transformation enables the linear regression model to fit more complex patterns within the data, such as quadratic relationships, which can significantly enhance the model's predictive capabilities. Furthermore, the use of polynomial regression may not be justified if the relationship between the variables is close to linear. In such cases, simpler models, such as linear regression, might not only suffice but also perform better due to their lower variance. This complexity can lead to overfitting, especially as the degree of the polynomial increases. Overfitting happens when a model learns not only the underlying pattern but also the noise in the training data to an extent that it negatively impacts the performance on new, unseen data. This is evident from the model performing poorly on the test dataset and is further reinforced by the degradation of R^2 as complexity increases.

3.5.8 Polynomial Regression (with Regularization)

Implemented a polynomial regression model with Ridge regularization to analyze our dataset. The use of polynomial features allows us to capture the non-linear relationships between the predictors and the response variable. Specifically, we expanded the feature set to include quadratic terms, which provides a more nuanced model of the data complexities than a simple linear approach. The regularization aspect of the model is handled through Ridge regression, which introduces an L2 penalty based on the alpha parameter. We set alpha to 0.01, balancing the need to control for overfitting while still allowing the model to fit the data sufficiently. This regularization helps to mitigate the potential overfitting issues commonly associated with polynomial regression, particularly when higher-degree terms are involved.

3.5.9 Multilayer Perceptron

We additionally employed a Multilayer Perceptron (MLP) model to tackle a regression task, aiming to predict continuous values based on input features. The MLP model, implemented using TensorFlow and Keras libraries, offered a versatile architecture suitable for capturing complex relationships within the data. The MLP architecture consisted of three dense layers, with 64, 32, and 1 neurons respectively. ReLU activation functions were applied to the hidden layers to introduce non-linearity, while the output layer remained activation-free, aligning with the regression nature of the task. For deciding the number of layers, we heuristically checked the model performance with upto 5 layers and noticed that thought the model converged faster the accuracy stayed the same as it had with 2 hidden layers. To ensure robust model performance, we divided the dataset into training and testing sets using the `train_test_split` function, with a test size of 0.2. This partitioning facilitated unbiased evaluation of the model's generalization capability. Model training involved the selection of hyperparameters, such as the number of epochs and batch size, alongside architectural parameters. We opted for the Adam optimizer and mean squared error (MSE) as the loss function for model compilation, with MSE serving as the evaluation metric during training. The MLP model underwent training for 30 epochs, with a batch size of 32, to iteratively adjust its parameters and minimize the loss function. Throughout the training process, we visualized the training and validation loss over epochs, providing insights into the model's convergence and generalization performance. Following training, we evaluated the trained MLP model's performance on the unseen test data. This evaluation included computing the test loss and MSE, offering quantitative measures of

the model's predictive accuracy. Additionally, we visualized the predicted versus actual values on the test data, allowing for a qualitative assessment of the model's performance

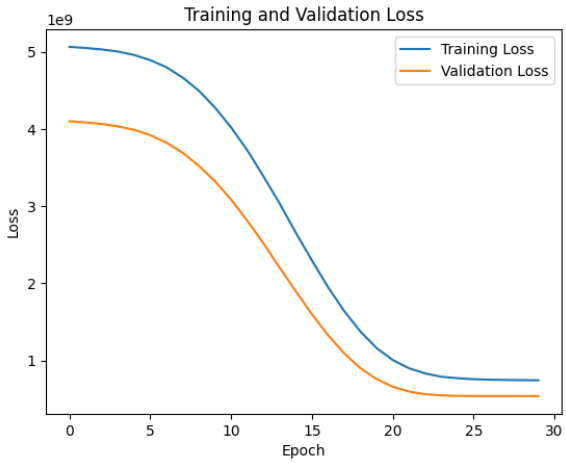

Figure 15: MLP: Training and Validation loss over the epochs

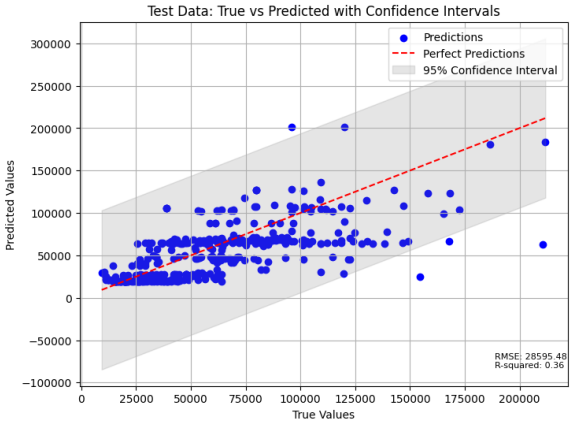

Figure 16: MLP: Actual vs Predicted Values

3.5.10 Multilayer Perceptron with Regularization

In our project, we expanded upon the Multilayer Perceptron (MLP) model by incorporating L2 regularization, a technique aimed at mitigating overfitting by penalizing large weights in the model. The addition of L2 regularization enhances the model's generalization capability and robustness to unseen data, offering improved performance in scenarios with high-dimensional feature spaces. The architecture of the MLP model with L2 regularization closely resembled the previous model, consisting of three dense layers with 64, 32, and 1 neurons respectively. However, the key distinction lies in the inclusion of L2 regularization for the dense layers, achieved through the `kernel_regularizer` parameter in Keras. This regularization term introduces a penalty proportional to the squared magnitude of the weights, thereby encouraging smaller weight values and reducing model complexity. The incorporation of L2 regularization distinguishes this approach from the previous one, offering additional control over model complexity and regularization strength. By penalizing large weights in the model, L2 regularization helps prevent overfitting and enhances the model's ability to generalize to unseen data.

Consequently, the MLP model with L2 regularization may exhibit improved performance, particularly in scenarios with high-dimensional feature spaces or limited training data.

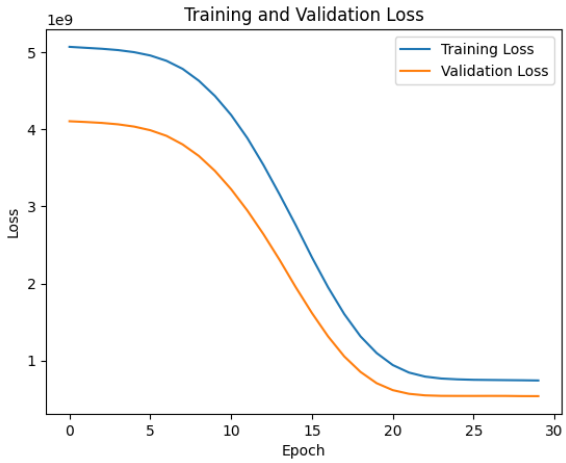

Figure 17: MLP with Regression: Training and Validation loss over the epochs

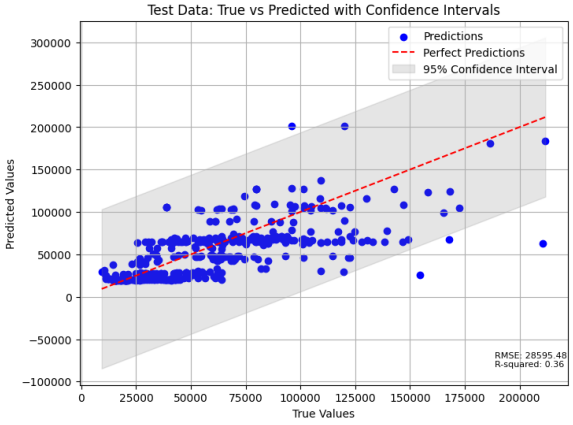

Figure 18: MLP with regression: Actual vs Predicted Values

If you used any **models or algorithms that are outside the scope of EE 559**, then give a more complete description of what they are. If the description would be long, you can give a summary of it in your report and cite reference(s) that give more complete descriptions. **State the parameters of the model and how they were chosen.** If a parameter is chosen by heuristics, state so. If a parameter is chosen by some model selection, optimization, or validation process, state so and describe the method. **Consider and describe degrees of freedom with the number of constraints** or data points you have. If you have **sets of results to show for this ML method**, include them here. (For a comparison of results from different ML methods, use the next subsection.) **Include any analysis and interpretation** of these (intermediate) results here.

Notes on presentation:

- **You do not have to present in-detail intermediary results for each method.** For example, no need to show cross-validation performance for each value of each tested parameter. However, be sure to state the range of values that was tested, the increment or number of values tried, along with the final parameter choice.

- **Do not copy and paste print-screen (or screenshots)** of results. Include numerical results in tables or as part of the text; save images as files and import or insert them into the report.

3.5.11 Baseline System Description

First Baseline System:

The baseline system utilized for regression tasks is the 1-Nearest Neighbor (1NN) Regression Algorithm. This choice is guided by its simplicity and effectiveness as a non-parametric method, making it a suitable starting point for comparison and benchmarking. The algorithm operates by predicting the target value of a test data point based on the target value of its nearest neighbor in the training set. No specific adaptation or parameter tuning is performed, ensuring consistency and comparability across evaluations. Evaluation metrics such as mean squared error (MSE), mean absolute error (MAE), or coefficient of determination (R-squared) are used to assess its performance.

Second Baseline System:

The Linear Regression model served as the second baseline for regression tasks in our study due to its simplicity and interpretability. Trained on the entire training set, it estimated coefficients using least squares and made predictions on the test set. Evaluation was based on standard regression metrics like Mean Squared Error (MSE) and Mean Absolute Error (MAE). The model required no hyperparameter tuning and served as a benchmark for comparing more complex algorithms. Remarkably, it yielded amazing results, highlighting its effectiveness even in scenarios where the relationship between features and target variables was relatively linear. This success underscores the importance of considering simpler models alongside more complex ones, as they can often provide competitive performance with lower computational overhead and greater interpretability.

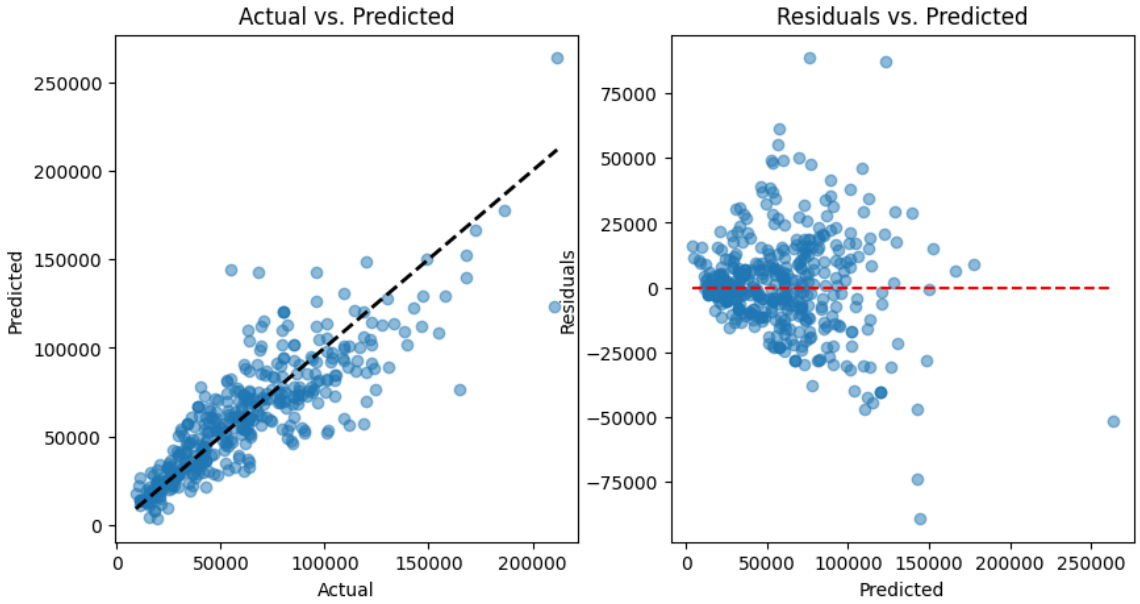

Figure 19: Linear Regression Performance

Model	Validation Dataset	Test Set	Comments
1-Nearest Neighbor (1NN) Regression	RMSE: 0.322, MAE: 0.231, R2: 0.753	RMSE: 0.340, MAE: 0.255, R2: 0.706	Achieves low RMSE and MAE values on both validation and test sets
Linear Regression	RMSE: 0.221, MAE: 0.175, R2: 0.884	RMSE: 0.277, MAE: 0.208, R2: 0.806	The feature engineering on the data lends good linearity to the data spread. Simple Linear regression had an R2 score of 0.88
Linear Regression Regularized (Best alpha= 0.01)	RMSE: 0.221, MAE: 0.175, R2: 0.884	RMSE: 0.277, MAE: 0.208, R2: 0.806	Similar performance to basic linear regression, indicating minimal impact of regularization
PCA + Linear Regression with L2 Regularization (Best Alpha = 1e-15)	RMSE: 0.238, MAE: 0.195, R2: 0.865	RMSE: 0.301, MAE: 0.230, R2: 0.769	Utilizing PCA for dimensionality reduction slightly increases RMSE and MAE
SVR Algorithm - Linear Kernel	RMSE: 0.237, MAE: 0.186, R2: 0.866	RMSE: 0.290, MAE: 0.219, R2: 0.786	Leverages linearity; Performs as well as the baseline models
PCA + SVR Algorithm (RBF Kernel) + Linear Regression (regularized)	RMSE: 0.272, MAE: 0.212, R2: 0.801	RMSE: 0.299, MAE: 0.229, R2: 0.773	Dimensionality reduction improves performance
Bayesian Multivariate Linear Regression	RMSE: 0.347, MAE: 0.243, R2: 0.688	RMSE: 0.449, MAE: 0.281, R2: 0.488	Good results but does not perform better than baseline models
PCA and Bayesian Linear Regression	RMSE: 0.249, MAE: 0.197, R2: 0.839	RMSE: 0.303, MAE: 0.230, R2: 0.767	Model performs well but does not outperform normal bayesian without dimensionality reduction
Polynomial Regression	RMSE: 0.917, MAE: 0.386, R2: -0.99	RMSE: 173988454673, MAE: 31142874085.2, R2: -7.19	Overfits after PCA, needs regularization to improve results; Performs better before dimensionality reduction
Polynomial Regression (with Regularization)	RMSE: 5.278, MAE: 1.095, R2: -65.203	RMSE: 7.147, MAE: 3.954, R2: -128.615	Despite regularization, the model exhibits high RMSE and MAE values
Multilayer Perceptron	RMSE: 2.145, MAE: 1.368, R2: -1977.975	RMSE: 1.979, MAE: 1.356, R2: -3995.685	The RMSE and MAE values are relatively low, indicating good predictive accuracy; R2 scores indicate poor generalization
Multilayer Perceptron with Regularization	RMSE: 2.122, MAE: 1.366, R2: -1935.119	RMSE: 2.129, MAE: 1.458, R2: -4620.652	Regularization does not help avoid overfitting

Table 2: Comprehensive Model Performance Analysis

4 Libraries Used and Custom Code Implementation

In this project, we utilized a comprehensive suite of Python libraries to facilitate various stages of data processing, analysis, and model development:

1. **NumPy and Pandas**: Employed for data manipulation and preprocessing tasks.
2. **Matplotlib and Seaborn**: Used for generating plots and statistical graphics for exploratory data analysis and model evaluation.
3. **Scikit-learn**: Offers various machine learning algorithms, including support vector machines and k-means, and tools for data preprocessing such as OneHotEncoder and StandardScaler.
4. **PrettyTable**: Used to organize output data into visually appealing tables, enhancing the presentation of results.
5. **Google Colab and Google Drive**: Provided a cloud-based environment for executing Python notebooks and storing data, facilitating easy collaboration.
6. **KFold and Train/Test Split**: These functions from Scikit-learn's model selection module were used for dividing data into training and testing sets and conducting robust cross-validation.

Although not utilized in this project, TensorFlow Probability is recognized for its capabilities in probabilistic reasoning and statistical analysis, which are particularly useful in Bayesian methodologies.

4.1 Custom Implementations

We developed several algorithms and functions specifically tailored to meet the project's requirements:

1. **Performance Metrics (RMSE, MAE, R^2 Score)**: The performance metrics—Mean Absolute Error (MAE), R-squared (R^2), and Root Mean Squared Error (RMSE)—were manually implemented using their respective mathematical formulas with the assistance of the NumPy library.
2. **1-Nearest Neighbor Regression (1NN)**: This algorithm estimates target values by calculating Euclidean distances between each test point and all training points. The nearest training point's target value is used for predictions. It underwent evaluation on both validation and test datasets.
3. **Ridge Regression with Alpha Optimization**: Implemented a function, `best_alpha`, to identify the optimal alpha for Ridge regression. This function uses cross-validation to minimize the RMSE, thereby preventing overfitting.
4. **Radial Basis Function (RBF)**: Developed functions to optimize RBF regression by determining the best gamma values. This involved a model selection technique using 5-fold cross-validation, covering a comprehensive range of gamma values to ensure thorough parameter tuning.
5. **Radial Basis Function (RBF) using Kmeans**: Further functions were developed to optimize RBF regression by finding the best combination of K and gamma using k-means clustering, demonstrating a strategic approach to parameter optimization.
6. **Bayesian Linear Regression**: This method includes several key steps:
   - *Initial OLS Estimation*: Initial estimates for the beta coefficients are computed using the normal equation. Alpha is estimated as the mean difference between observed and predicted values by the initial model.
   - *Bayesian Update*: Bayesian updating is applied, assuming that both the prior and likelihood distributions are normally distributed. This step involves calculating the posterior covariance and mean for the beta coefficients.
   - *Predictions*: Predictions are made for the training data using the posterior mean estimates of the coefficients and alpha.

5 Contributions of Each Team Member

The project was a collaborative effort, with each team member focusing on different aspects crucial for its success. Below is a breakdown of contributions made by each team member:

5.1 Sumukh

- **Initial Data Study:** Identified potential categorical data columns for further analysis and processing.
- **Creating Correlation Matrix and Its Underlying Study:** Developed and analyzed the correlation matrix to identify significant relationships between features.
- **Model Implementation:**
  - Implemented and tested several machine learning models including:
    1. 1-Nearest Neighbor Regression (1NN)
    2. Linear Regression
    3. Support Vector Regression (SVR) with both linear kernel and grid search optimization
    4. Multi-Layer Perceptron (MLP) with and without regularization

5.2 Tanvi

- **Pre-processing, Feature Engineering, and Exploratory Data Analysis (EDA):**
  - Applied one-hot encoding to categorical variables.
  - Performed data splitting and engineered new features like Pixels Per Inch (PPI).
  - Conducted extensive EDA including plotting to visualize data characteristics and interactions.
- **Model Implementation:**
  - Developed and optimized advanced models including:
    1. Ridge Regression
    2. Radial Basis Function (RBF) with Ridge Regression
    3. RBF using K-means with Linear Regression
    4. Polynomial Regression with and without regularization
    5. Bayesian Regression
    6. Principal Component Analysis (PCA) for dimensionality reduction

5.3 Joint Contributions

- **Report Writing:** Both Tanvi and Sumukh contributed to compiling, writing, and revising the final project report, ensuring a comprehensive and cohesive presentation of the project outcomes.

6 Summary and conclusions

1. Regularized Linear Regression The improvement in R^2 and reduction in best alpha suggest that PCA was effective in extracting the most informative features from the data, thus simplifying the model without losing predictive power. After PCA, the best alpha drastically reduced to 1e-15 from 0.01 which is effectively zero, indicating almost no regularization is needed. 2. Polynomial Regression After PCA and extensive feature engineering, the data exhibited almost linear characteristics, which

led to underwhelming performance when using polynomial regression. However, the performance significantly improved with regularized polynomial regression as the regularization term alpha was increased. This improvement suggests that the slight complexity present in the data was effectively captured by increasing the regularization, which helped to fine-tune the balance between model complexity and fitting accuracy

6.1 Summary:

The evaluation encompassed a range of regression models, including linear regression, polynomial regression with regularization, SVR, and Bayesian multivariate linear regression. Key metrics such as RMSE, MAE, and R2 scores were employed to assess model performance. Additionally, dimensionality reduction techniques like PCA were applied to enhance model efficacy, particularly when combined with SVR and regularized linear regression. Overall, models leveraging dimensionality reduction alongside SVR and regularized linear regression demonstrated superior performance, underscored by their high R2 scores and effective utilization.

6.2 Conclusion:

While dimensionality reduction techniques coupled with SVR and regularized linear regression proved advantageous, certain models exhibited shortcomings. Polynomial regression with regularization, for instance, suffered from severe overfitting, rendering it ineffective. Similarly, Bayesian multivariate linear regression yielded satisfactory results but failed to surpass baseline models. These findings underscore the critical importance of thoughtful model selection and the potential benefits of integrating dimensionality reduction methods into regression analysis, paving the way for more robust and accurate predictive modeling in future endeavors.

References

[1] *Alpha Tuning*, `https://stats.stackexchange.com/questions/166950/alpha-parameter-in-ridge-regression-is-high/`

[2] *Ridge Regression Tutorial*, `https://www.analyticsvidhya.com/blog/2016/01/ridge-lasso-regression-python-complete-tutorial/`

[3] *RBF Network*, `http://www.di.unito.it/~botta/didattica/RBF_1.pdf`

[4] *RBF Overview*, `https://faculty.ist.psu.edu/vhonavar/Courses/ds310/rbf.pdf`