Forecasting Laptop Prices Using Regression Models

EE 559 Course Project
Data Set: Laptop Prices

Khushboo Khatri (khatrik@usc.edu), Neha Vedam (nvedam@usc.edu)

April 26, 2024

1 Abstract

In this project, our objective is to identify the most effective regression model for predicting laptop prices based on the provided dataset. Initially, we focused on refining preprocessing and feature selection techniques to ensure a comprehensive representation of the dataset's patterns. Enhanced features led to the most promising experimental outcomes during cross-validation, particularly with support vector regression (RBF kernel) with linear regression, as well as Random Forest, achieving accuracies of 90.56% and 83.82%, respectively. Upon evaluation on the test data, these models demonstrated the highest test accuracies, recording 74.69% for SVR and 74.42% for Random Forest. The RMSE and MAE values reported by these selected models were among the lowest observed in our experiments.

2 Introduction

2.1 Problem Assessment and Goals

Our project utilizes a dataset which includes key features such as the brand, processor type, RAM capacity, storage details, and display specifications among others. This dataset has been provided to ensure consistency and control over the variables involved in our study.

The primary goal of our project is to develop a predictive model that can accurately estimate the market prices of laptops based on their specifications. This involves applying machine learning regression techniques to understand and quantify the relationship between laptop features and their pricing. The objectives of this project are:

1. Data Handling: To prepare the data for modeling, we'll conduct a series of preprocessing steps. This includes standardizing numerical data, encoding categorical variables, reducing dimensionality with help of different methods. This phase ensures that the data is clean and formatted in a way that maximizes the performance of the machine learning algorithms.
2. Model Development: To implement and evaluate multiple machine learning models to find the most effective one for predicting laptop prices. As per the project requirements, we will explore at least four different machine learning models, including non-probabilistic models, support vector regression, neural networks, and probabilistic models. This diverse array of models will help us assess which model adapts best to the intricacies of laptop pricing.
3. Performance Evaluation: To rigorously assess the performance of these models using various metrics like R2 score, MAE, and RMSE. We aim to establish a robust model by comparing a baseline system, a trivial system, and our chosen models to identify significant improvements and performance enhancements.

3 Approach and Implementation

3.1 Dataset Usage

We utilized a dataset comprising 901 data points. Each data point started with 11 features. Preprocessing and exploratory data analysis allowed us to expand these to 84 features, which included the desired feature 'Price'. We implemented the proper preprocessing techniques after carefully analyzing important categorical features like GPU and CPU, which showed high cardinality. One-hot encoding was used to encode categorical characteristics in the dataset in order to make model training easier.

We further did 10 fold cross-validation. The validation results were averaged to minimize the impact of anomalous data splits and guarantee robustness. This thorough cross-validation architecture helped our feature selection procedure as well. In order to determine which characteristics were the most predictive, we used strategies like Backward Feature Elimination and Univariate Feature Selection. This allowed us to ensure that the features we chose were consistent across several approaches.

The validation set was crucial in tuning our model and selecting features, ensuring that the selections were reflective of generalizable performance rather than fitting to specific data quirks. To ensure that it was utilized sparingly and prevent biases in model assessment, the test set was only used after the model was finalized.

3.2 Exploratory Data Analysis

In this section, we examine the exploratory data analysis (EDA) techniques employed to extract the primary characteristics of the dataset. EDA serves as an initial phase in data analysis, aiming to uncover the dataset's underlying structure, detect outliers, and identify patterns and relationships. We have utilized visual methods - bar plots as shown in figure - which have helped to visualize the distribution and frequency of data across various categories which has facilitated the identification of any anomalies and trends within the dataset. Analysis based on plots for each feature has been discussed below.

The **Company** feature shows a significant imbalance, with certain companies being overrepresented. This suggests that certain laptop brands are more common in the dataset, which may influence the model's performance.

TypeName indicates a categorical variable with a clear dominance of 'Notebook' type, followed by 'Ultrabook' and 'Gaming' laptops. One-hot encoding of this variable is justified to convert these categories into a format suitable for modeling. Given the imbalance, models may require tuning for class distribution sensitivity.

ScreenResolution feature has a wide range of unique values with varying frequencies, with some resolutions being much more common than others. Feature Engineering steps could involve grouping similar resolutions to reduce dimensionality or one-hot encoding to capture the variance across resolutions.

The **Memory** feature distribution is quite varied, with certain memory sizes being more frequent. One might consider grouping similar memory configurations to reduce complexity or applying one-hot encoding. Feature engineering could also be used to separate memory type (SSD, HDD) from size to see if that impacts the model's predictive capability.

OpSys is highly imbalanced, with 'Windows 10' being the most frequent operating system by a large margin.

Cpu has a high number of unique categories, with certain processors appearing much more frequently than others.Due to the high cardinality, it may be wise to aggregate similar CPUs or to encode them into broader categories (e.g., Core i7, Core i5) to reduce sparsity from one-hot encoding.

Gpu also exhibits high cardinality with certain GPUs appearing frequently.Feature engineering could steps might include categorizing GPUs by performance tier or manufacturer to avoid creating too many sparse features through one-hot encoding.

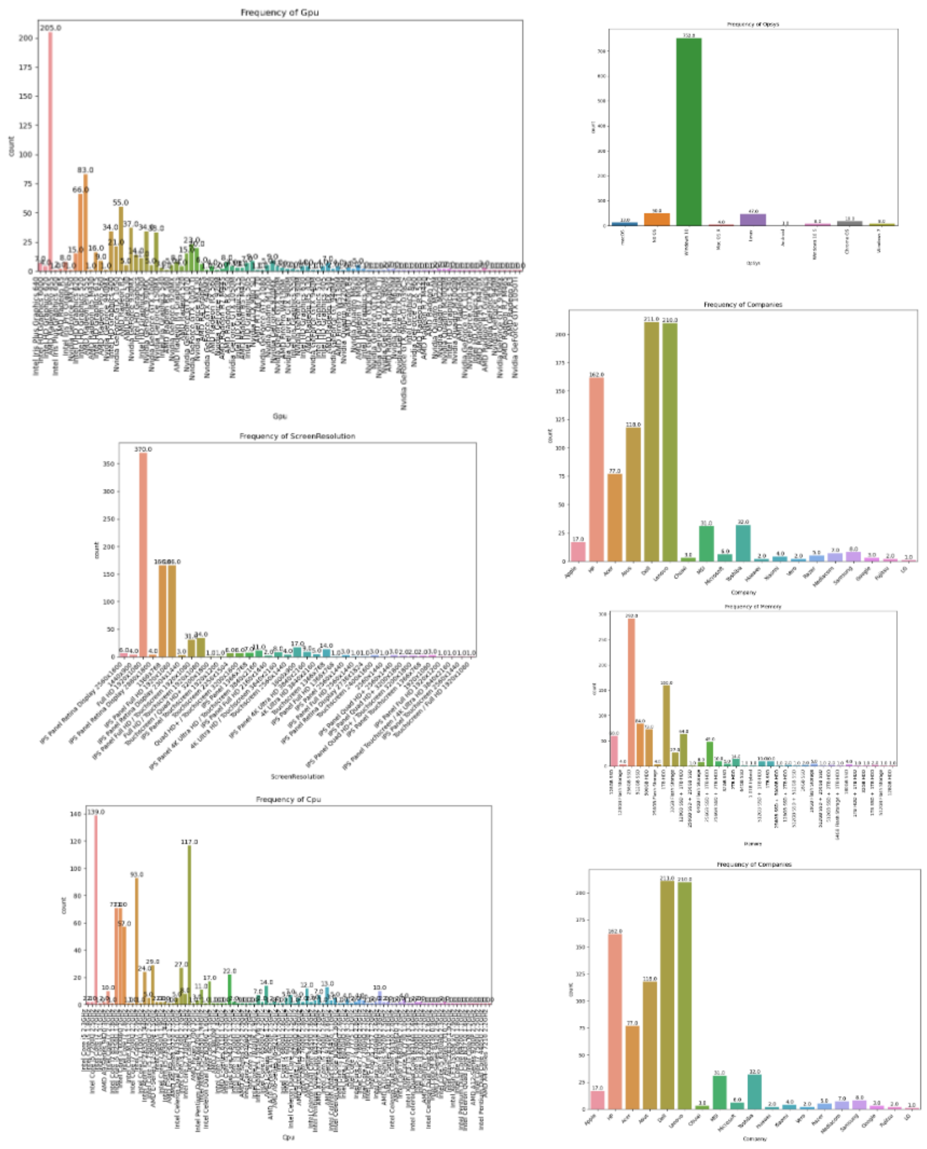

Figure 1: EDA Plots

3.3 Preprocessing

We changed some of the columns that included units such as ‘GB’ and ‘kg’ by removing these units to make them completely numerical. We also removed the unit ‘GHz’ from cpu_speed column, converting it and casting it from string to float numeric value.

The lambda function is used to determine the unique count of the column Number, through which we were able to conclude that they are serial numbers which can be dropped without affecting the output. Additionally we used libraries to determine if any of the features have null values. Some additional pre-processing methods:

3.3.1 Standardization:

Feature standardization, often known as z-score normalization, is a vital preprocessing step in data analysis, particularly for algorithms that assume data is centered around zero and with a standard deviation of one. This technique modifies the data such that the features have the properties of a standard normal distribution with a mean of zero and a standard deviation of one.

The transformation is performed by subtracting the mean of each feature and then scaling it by the feature's standard deviation. The formula for standardization is as follows:

$$Z = \frac{(X - \mu)}{\sigma} \tag{1}$$

Where X is the original feature value, μ is the mean of the feature values, and σ is the standard deviation of the feature values.

After applying standardization to our dataset, we noticed while some models benefited from standardization, others either didn't improve or worsens. We used case by case setting of applying standardization for each model. This lack of improvement could be attributed to several factors: the model may inherently be robust to different feature scales, the data could already be distributed in a manner that standardization does not significantly alter, or the model's performance may be predominantly influenced by other data characteristics that standardization does not address.

3.3.2 Skewness:

Skewness is a measure of the asymmetry of the probability distribution of a real-valued random variable about its mean. In a perfectly symmetrical distribution, the mean and the median are the same, resulting in a skewness of zero. When a distribution is skewed to the right, it is said to be positively skewed, meaning that the right tail of the distribution is longer or fatter than the left. Conversely, if a distribution is skewed to the left, with the left tail longer or fatter than the right, it is negatively skewed. Skewness can significantly affect statistical analyses and model performance as many algorithms assume that the underlying data is normally distributed.

We used logarithmic transformation, which can stretch out values on the left and compress the long tail on the right, for data that was positively skewed. This lessens the impact of high numbers and helps to center the majority of the data. However, a cubic transformation was used for data that was negatively skewed. In order to create a more symmetric distribution, this entails cubing each data point, which has the effect of stretching the distribution out on the left side and compressing it on the right. While complete data normalization is not a given with these modifications, they can greatly lessen skewness, increasing the accuracy and dependability of statistical models.

In our project, we observed that addressing skewness as a part of the preprocessing step enhanced the performance of certain models. However, for others, the improvement in validation accuracy was marginal. Consequently, we have tailored our approach and handled these instances on a case-by-case basis.

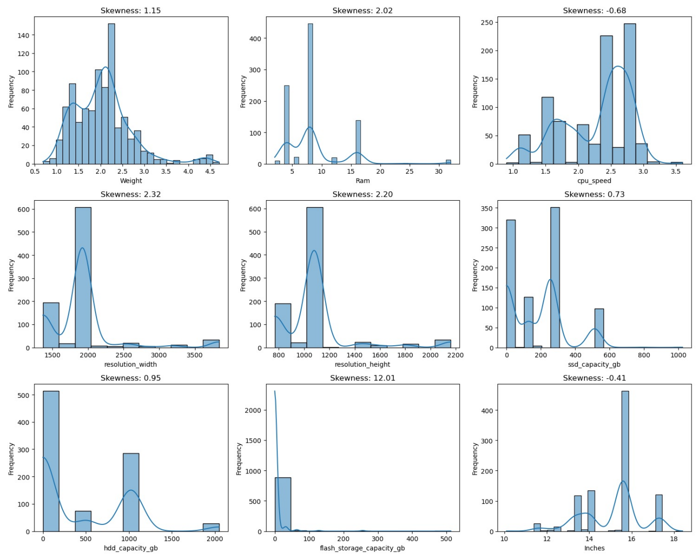

Figure 2: Distribution and Skewness Analysis of Numeric Features

3.4 Feature Engineering

GPU Differentiation:
To better capture the impact of GPU performance on laptop pricing, GPUs were categorized as 'Integrated' or 'Discrete'. Integrated GPUs, commonly from Intel, are known for their on-chip integration and shared memory with the CPU. Discrete GPUs, typically from Nvidia and AMD, boast dedicated hardware and memory, providing advanced graphics performance. This distinction was encoded based on brand and model identifiers.

CPU Details Extraction:
The CPU information was granularly parsed into brand, model, and operational speed. This parsing allowed the models to potentially discern the impact of CPU characteristics on laptop pricing.

Screen Resolution Parsing:
Resolution details were dissected to obtain width, height, and whether the screen was of IPS type, had Retina display, or featured touchscreen capabilities. These factors can significantly influence user preference and pricing.

Memory Composition:
The memory configuration was meticulously broken down into SSD, HDD, and flash storage capacities, including an indicator for hybrid storage systems. This provided a detailed look at storage options, which are a vital consideration for pricing laptops.

Encoding Categorical Features:
To transform categorical data into a machine-readable format, one-hot encoding was applied to various features like 'Company', 'TypeName', and 'OpSys', among others. This was essential to prepare the dataset for modeling, as machine learning algorithms generally require numerical input.

Data Consolidation:
After applying ColumnTransformer for one-hot encoding, the resulting sparse matrix was converted into a readable DataFrame, and new feature names were generated to reflect the transformed dataset.

Final Dataset Preparation:
Finally, 'cpu_speed' was converted from a string to a floating-point number for better computational

handling, and all newly engineered features, along with the pre-existing ones, were collated into a final DataFrame ready for model training and evaluation.

3.5 Feature Dimensionality Adjustment or Feature Selection

To eliminate any redundant or unnecessary features that could impair model performance, feature dimensionality adjustment is carried out. The processed dataset should follow the condition, N > (3-10)* d.o.f., where N stands for total number of datapoints or constraints on weights and d.o.f. stands for degrees of freedom. After feature engineering, this equation was no longer valid in our case. The model would then learn the train set's noise and data pattern, which could result in overfitting.Additionally, redundant or processed features like 'ScreenResolution', 'Cpu', 'Memory', and 'Gpu' were dropped to avoid duplication of information. We experimented with the following feature selection methods.

3.5.1 Univariate Feature Selection – ANOVA

Univariate feature selection (Analysis of Variance) is a technique that assesses the individual predictive power of each feature by comparing the means of groups formed based on these features. This method helps to identify variables that have statistically significant impacts on the dependent variable, allowing for the reduction of the feature set to those most relevant for modeling. ANOVA calculates an F-statistic which is a ratio of the variance between the groups to the variance within the groups. A higher F-statistic suggests that the group means are not all equal and that at least one group mean significantly differs from the others.

3.5.2 Univariate Feature Selection – Pearson Correlation Coefficient

UFS - Pearson Correlation Coefficient (PCC) involves measuring the linear correlation between each independent variable and the target variable by quantifying both the strength and direction of this relationship. Features with a high absolute value of PCC are considered strong predictors and are typically retained for model building. These values are between -1 and +1, where +1 indicates a perfect positive linear relationship, -1 indicates a perfect negative linear relationship, and 0 suggests no linear correlation. This coefficient is calculated by dividing the covariance of the two variables by the product of their standard deviations.

3.5.3 Backward Sequential Feature Selection

Backward Sequential Feature Selection (SFS) is a technique that starts with the full set of features and iteratively removes the least significant feature at each step, based on a specific model criterion. This process continues until a predetermined number of features is reached or the removal no longer improves the model's performance. The aim is to identify a subset of features that optimizes model performance, while reducing complexity and potential overfitting by eliminating irrelevant features. The reason we have used backward SFS is because while Forward SFS is often better for smaller datasets or when fewer features are needed, Backward SFS can be more efficient with datasets that start with many features. Although the drawback is that it is computationally more intensive due to the larger initial model.

Despite obtaining nearly identical features and validation scores across these methods, we observed slightly superior performance with PCC and proceeded with the features identified through this approach.

For each model, we identified the optimal features based on their performance as measured by PCC in validation scores (plot shown below for Baseline, for other models can be seen in code file). These scores were averaged over 10-fold cross-validation to ensure robustness and generalizability.

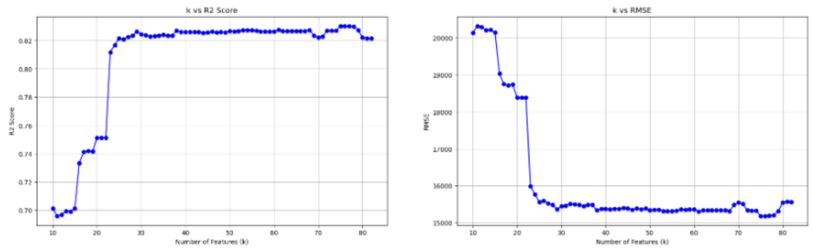

Figure 3: K features vs R2 score and K features vs RMSE

3.6 Training, Classification or Regression, and Model Selection

In the section we will describe the machine learning (regression) models that were used.

The degrees of freedom of the data provided to each model was chosen using the feature selection (Univariate Feature Selection) and is detailed for each model in the table[1] below. The results of the validation scores are also described in the same table.

1. **Trivial System:** We have created a model that always outputs the mean of target value (of training set).

2. **Baseline System:** We used a basic Linear Regression model with default hyper parameters and no regularization.Linear regression is a statistical method that models the relationship between a dependent variable and one or more independent variables by fitting a linear equation to observed data. The key objective of linear regression is to predict the value of the dependent variable based on the linear combination of independent variables, minimizing the sum of the squares of the differences between the observed and predicted values.

3. **Lasso Regression:** Least Absolute Shrinkage and Selection Operator regression, is a type of linear regression that includes a regularization term. This term is the sum of the absolute values of the coefficients, multiplied by a tuning parameter λ. The objective of the lasso regression is to minimize the quantitiy:

$$\text{RSS} + \lambda \sum_{j=1}^{p} |\beta_j| \tag{2}$$

   where RSS (Residual Sum of Squares) is $\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2$ and β_j are the coefficients, and λ is a non-negative parameter that controls the strength of the penalty. This penalty encourages sparsity in the coefficients, potentially setting some to zero, thus performing variable selection.

   We have chosen Lasso for this dataset as it is particularly useful when dealing with datasets that have a large number of features, some of which may be correlated or irrelevant. By penalizing the absolute value of the coefficients, it helps in reducing overfitting by shrinking less important feature coefficients to zero, effectively removing them from the model. This results in a model that is simpler and more interpretable while maintaining good predictive performance. The choice of λ is crucial and is typically selected via cross-validation to balance model complexity and fitting accuracy. If $\lambda = 0$, lasso regression reduces to ordinary least squares regression. As λ increases, more coefficients are driven to zero, leading to a sparser model.

   We have chosen $\lambda = 0.0005$. This is very low and almost approximate to linear regression. However, increasing it hurts the model performance.

4. **Ridge Regression:** Ridge regression is a technique used to analyze multiple regression data that suffer from multi-collinearity. It includes a regularization term that penalizes the square

of the coefficients, thereby shrinking them towards zero. The objective function to minimize in ridge regression is:

$$\text{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{3}$$

where RSS (Residual Sum of Squares) is $\sum_{i=1}^{n}(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2$, β_j are the coefficients, and λ is a non-negative parameter controlling the penalty strength. This squared penalty discourages large coefficients and helps to reduce model variance without substantially increasing bias.

We have chosen Ridge regression as it is particularly beneficial for datasets with a large number of features because it handles multi-collinearity (which can arise from increased number of features) effectively by distributing the coefficient values more evenly across all features, compared to ordinary least squares which might overfit in such scenarios.

For ridge, we found that $\lambda = 1$ was the best hyperparameter value based on heuristics.

5. **Polynomial Regression:** It is a form of regression analysis in which the relationship between the independent variable x and the dependent variable y is modeled as an nth degree polynomial. The general form of the polynomial regression model is:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n + \epsilon \tag{4}$$

where $\beta_0, \beta_1, \ldots, \beta_n$ are the coefficients, and ϵ is the error term. This approach allows the model to capture more nuanced relationship between the variables as compared to simple linear regression.
We wanted to show how this type of regression can be particularly challenging with large datasets featuring many variables, as the number of terms quickly increases with the degree of the polynomial.

This potentially leads to high computational costs. We found degree = 2 , polynomial feature was already causing overfitting issue and increasing degree any further would not help this and it was also not possible to fit in our laptop's memory.

6. **Support Vector Regression (RBF Kernel):** In SVR, the goal is not only to fit the data but to do so while maintaining a balance between model complexity and the amount to which deviations larger than a specified threshold are tolerated. The RBF kernel is a popular choice for SVR as it allows the model to handle non-linear relationships by mapping input features into higher-dimensional spaces where a linear regression surface may be fitted.

The mathematical formula of SVR with an RBF kernel is:

$$y = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(x_i, x) + b \tag{5}$$

where x_i are the support vectors, α_i and α_i^* are the dual coefficients, b is the bias term, and K is the RBF kernel function defined as:

$$K(x_i, x_j) = \exp\left(-\gamma \|x_i - x_j\|^2\right) \tag{6}$$

Here, γ is a parameter that defines the spread of the kernel and therefore controls the degree of non-linearity of the decision surface.

We used an SVR with RBF kernel to highlight on the fact that it faces challenges when applied to datasets with high number of features. This is because the RBF kernel maps input features into a higher-dimensional space which can significantly increase the complexity of the model when there are higher number of features. Moreover, it is highly dependent on fine-tuning hyperparameters which can be challenging in these cases.

We selected kernel = rbf (Radial Basis Function), C=100, epsilon = 0.12, gamma = 0.001 based on heuristics.

7. **Support Vector Regression (RBF Kernel) followed by Linear Regression:** This combined approach using SVR with a RBF kernel followed by Linear Regression is used to effectively address complex datasets with nonlinear patterns initially and linear relationships subsequently. The process can be summarized with these steps:

   - *SVR with RBF Kernel:* Initially, SVR with an RBF kernel is used to model complex, nonlinear relationships in the data. The RBF kernel transforms the data into a higher-dimensional space where linear relationships are more discernible, making it possible to capture intricate patterns that simple models might miss. This step involves tuning hyper-parameters like the penalty parameter C and the kernel coefficient γ to optimize performance.
   - *Feature Transformation:* The outputs or the learned features from the SVR model are then used as inputs for the next stage. This step often involves using the decisions or the distances from the hyperplane generated by the SVR as new features, which encapsulate the nonlinear interactions in the original data.
   - *Linear Regression:* Following the feature transformation, Linear Regression is applied. This model can effectively interpret the transformed, now more linearly related features, providing a straightforward understanding of how these features impact the target variable. The simplicity of Linear Regression also helps in understanding the influence of each feature and provides clear model interpretation.

   We wanted to highlight the strength of using this hybrid modeling approach as it is particularly useful in scenarios where the data exhibits complex patterns initially but can be linearly described after certain transformations. This combination can lead to improved prediction accuracy and model interpretability compared to using either model alone.

   We used the same hyper-parameters described above in model 6 and coupled this with default linear regression model.

8. **Random Forest (Non EE559):** It is an ensemble learning method used for both classification and regression tasks. RF operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. The general principle of RF is to combine the predictions of several trees, built on different subsets of the dataset, to reduce the variance without increasing the bias significantly. The model creation process includes:

   - *Bootstrap Sampling:* For each tree in the forest, a random bootstrap sample from the original dataset is selected.
   - *Building Trees:* A decision tree is built for each bootstrap sample. The trees are grown to their maximum size without pruning, which typically helps in achieving the lowest bias.
   - *Random Feature Selection:* When splitting a node during the construction of the tree, rather than searching for the best split among all features, a random subset of features is selected, and the best split is found within this subset. This randomness helps in making the model more diverse and reduces the variance.
   - *Tree Growth:* Each tree is grown independently without any interaction between them during the building phase. Nodes are split using the best split among the randomly selected features until the tree reaches the maximum specified depth, or cannot be split further due to minimum node size.

- *Aggregation:* Once all trees are built, predictions are made by averaging the outputs (for regression problems) or by majority voting (for classification problems) across all trees. This aggregation helps to improve accuracy and control overfitting by averaging out biases.

We used Random Forests in this project to show how it handles high-dimensional spaces and large numbers of training examples well. It improves on the accuracy of single decision trees by reducing overfitting. However, this can become complex and require significant computational resources and time to train and predict. And are not easily interpretable compared to simpler models like decision trees, since the forest structure is quite complex.

We did grid search to find the best estimator. The best estimator had hyperparameters: max_depth = 20, min_samples_leaf = 1, min_sample_split = 2, n_estimators = 100

9. **Decision Tree (Non EE559):** Decision Trees are supervised learning algorithms used for classification and regression tasks. It operates by constructing a binary tree from the training dataset using a recursive partitioning approach that splits the data into subsets based on the feature that maximizes a specific objective function. This approach can be enumerated as:

- *Node Splitting:* Nodes are split based on the feature and threshold that maximize purity gain, typically using Gini impurity, defined as:

$$Gini = 1 - \sum_{i=1}^{k} p_i^2 \tag{7}$$

where p_i represents the probability of an object being assigned to class i. This metric quantifies the likelihood of incorrect classification if a random label from the subset were chosen according to the current distribution, with the objective to minimize this probability.

- *Recursive Splitting:* This purity-based criterion is recursively applied to each subset until a predefined stopping criterion is achieved. Criteria include a maximum depth of the tree, a minimum subset size at a node, or a minimal gain in purity from a potential split, enhancing computational efficiency and model accuracy.

- *Pruning:* Post-construction, the tree may be pruned by retrospectively removing splits that provide minimal contribution to predictive accuracy, effectively reducing model complexity and mitigating overfitting risks.

Through using a Decision Tree for this project, we wanted to show that this approach requires careful consideration to prevent overfitting. Due to its depth and complexity, decision trees can create overly complex models that perfectly fit the training data but perform poorly on unseen data. Pruning strategies and setting limits on the depth of the tree are essential techniques to ensure that the model generalizes well to new datasets.

We did grid search to find the best estimator. The best estimator had hyperparameters: max_depth = 10, min_samples_leaf = 1, min_sample_split = 10

10. **Bayes Ridge (Probabilistic- Non EE559):** Bayesian Ridge Regression is an approach to linear regression that incorporates Bayesian statistics, offering a probabilistic view of the model parameters. Unlike standard ridge regression which computes point estimates for coefficients with a penalty on their size, Bayesian Ridge Regression estimates a probability distribution for these coefficients.

The process begins by assuming a prior distribution over the coefficients (which is typically Gaussian):

$$\beta \sim \mathcal{N}(0, \alpha^{-1} I) \tag{8}$$

where α is the precision of the prior distribution, representing the strength of regularization. The likelihood of observing the data given the coefficients is also assumed to be Gaussian:

$$y \sim \mathcal{N}(X\beta, \lambda^{-1}I) \tag{9}$$

where λ is the precision of the noise. Bayesian Ridge Regression computes the posterior distribution of the coefficients given the data, which is also Gaussian due to the conjugacy between the Gaussian prior and likelihood. It maximizes the marginal likelihood of the observed data by adjusting α and λ, which are hyperparameters that control the model complexity and the noise level, respectively.

We wanted to show that Bayes Ridge can be effective for datsets with high number of features as by incorporating the prior about the distribution of the coefficients and optimizing hyperparameters, it naturally includes regularization, which helps prevent overfitting — a common issue in high-dimensional data spaces.

We chose hyperparameters: alpha_1 = 1e-6, alpha_1= 1e-6, lambda_1= = 1e-6, lambda_1= = 1e-6 based on heuristics.

11. **Artificial Neural Networks:** MLP is a type of feedforward artificial neural network that consists of multiple layers of neurons, typically including one input layer, several hidden layers, and one output layer. Each neuron in one layer connects to every neuron in the next layer, forming a densely connected structure. MLPs are capable of modeling complex nonlinear relationships in data by adjusting the weights of these connections through a process known as learning.

    The learning process in an ANN involves:

    - *Forward Propagat* Input data is fed into the input layer and passes through subsequent layers. At each neuron, a weighted sum of the inputs is computed and passed through a nonlinear activation function, such as ReLU or sigmoid, to produce the neuron's output.
    - *Backpropagation:* After obtaining the output, the network calculates the error in prediction. The error is then propagated back through the network, layer by layer, adjusting the weights to minimize the loss function, typically using gradient-based optimization methods like stochastic gradient descent.
    - *Iteration:* This process repeats over multiple iterations or epochs, with the network continually adjusting its weights to minimize the error on training data, thereby learning the optimal weights for predicting the target variable.

    In using ANN, we wanted to demonstrate that ANNs can perform well on high-featured datasets, especially when the relationships between features and outputs are complex and nonlinear. This is due to their architecture which includes multiple layers of neurons and proper training using backpropagation and optimization algorithms. Our model architecture has been shown in figure 4.

12. **K-Nearest Neighbors (Probabilistic):** K-Nearest Neighbors (KNN) Regression is a non-parametric method which is used in statistical and machine learning for predicting a continuous variable. KNN regression estimates the target variable by averaging the values of the k nearest neighbors to a query point, with proximity typically measured by Euclidean distance or other distance metrics. The main steps in KNN regression are:

    (a) *Choose the number of neighbors k*: The parameter k determines how many of the nearest training examples in the feature space are considered for predicting the output variable.

    (b) *Select a distance metric*: Common choices include Euclidean, Manhattan, or Minkowski distance. The distance metric helps to identify which data points are closest to the point of interest.

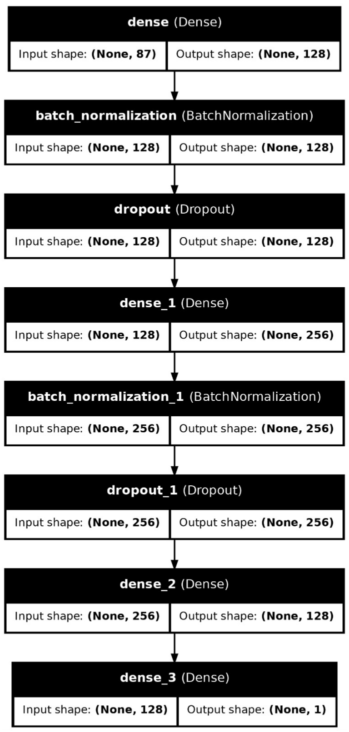

Figure 4: Artificial Neural Network Architecture

(c) *Calculate the distances*: For a given test sample, distances from all training samples are computed to find the nearest neighbors.

(d) *Aggregate the neighbors' outputs*: For regression, the prediction is typically the mean or median of the dependent variable of the nearest neighbors.

KNN regression can become computationally intensive as the dataset size increases, since distance calculations are required for each test instance.
We wanted to highlight how it is sensitive to the local structure of the data and can perform poorly if the feature space has many dimensions. Proper selection of k and the distance metric are crucial for good performance.

We did grid search to find the best estimator. The best estimator had hyperparameters: metric = manhattan, n_neighbors = 3, weights = distance

3.6.1 Metrics used for Model Selection and Comparison

In evaluating the predictive accuracy of our regression model, we employed the following statistical metrics: R-squared (R^2), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE).

1. **R-squared (R^2)** measures the proportion of variance in the dependent variable that is predictable from the independent variables. It is mathematically defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{10}$$

where y_i represents the observed values, $\hat{y}_i$ denotes the predicted values, and $\bar{y}$ is the mean of the observed data. R^2 ranges from 0 to 1, where a higher value indicates a better fit of the model to the data.

2. **Root Mean Squared Error (RMSE)** is the square root of the average of the squares of the errors. The RMSE is a measure of the accuracy of a model, indicating the average magnitude of the error. It is expressed as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{11}$$

where n is the number of observations. RMSE is always non-negative, and a lower value indicates a better fit.

3. **Mean Absolute Error (MAE)** is the average of the absolute errors, which are the distances between the predicted values and the actual values without considering the direction. MAE is given by the formula:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{12}$$

Similar to RMSE, a lower MAE value suggests a better model performance with less deviation from the actual values.

Each of these metrics provides a different perspective on the error characteristics of the model, helping in comprehensively understanding its performance across various aspects of the dataset. The number of features for each model's validation dataset were chosen to optimize the model performance and is recorded in the table as the parameter k.

Models	**Accuracy - R2 score**	**MAE**	**RMSE**	**k**
Baseline (Linear)	0.8344	12077.92	15888.33	-
Trivial	-0.00035	27372.76	36870.63	-
Lasso	0.8302	11884.29	15187.78	75
Ridge	0.8159	12076.59	15813.47	75
Polynomial	0.7517	12570.56	18366.79	12
Support Vector Regression (RBF Kernel)	0.0568	23931.99	35801.49	10
SVR + Linear	0.9056	8276.25	11885.83	68
Bayes Ridge (Probabilistic)	0.8101	11924.22	16062.39	29
Artificial Neural Network	0.8787	9527.93	13796	29
Decision Tree	0.7914	10610.59	16833.70	30
Random Forest	0.8382	9336.13	14825.88	42
K-Nearest Neighbors	0.7869	10466.03	17013.97	81

Table 1: Results of Machine Learning (Regression) Models on Validation set

4 Results and Analysis: Comparison and Interpretation

Based on the validation scores seen in table 1, we found 3 best models. These were:

1. SVR + Linear : For this model, we did the pre-processing, and Feature Engineering steps mentioned in above section. We did not used logarithmic transformation or standardization for this model. We chose degree of freedom of 68 based on our validation results of the model. The exact architecture and hyperparameters are the same as describe in section 3.6.

2. Artificial Neural Networks: For this model, we did the pre-processing, and Feature Engineering steps mentioned in above section. We also included standardization for this model. The exact architecture of the neural network including number of layers and their shape has been provided in section 3.6.

3. Random Forest (Non 559): For the specified model, we employed the preprocessing and feature engineering techniques outlined in the previous section. Logarithmic transformations and standardization were not incorporated into the preprocessing pipeline for this particular model. The selection of 42 degrees of freedom was determined by the outcomes observed during the model's validation phase. The precise architecture and hyperparameters align with those detailed in section 3.6 of our documentation.

These are our best performing models. The intricate balance between model complexity and predictive performance is evident in our results. Our best-performing system utilized a rbf kernel in SVR (transformation of dataset) followed by Linear Regression, which underscores the power of kernel methods in capturing underlying patterns without overcomplicating the model. This, paired with proper preprocessing and feature selection, provided a robust prediction mechanism as reflected in the test metrics.

Models	**Avg. Validation Accuracy - R2 score**	**RMSE**	**MAE**	**k**
Trivial	-0.00035	27372.76	36870.63	-
Baseline (Linear)	0.8344	12077.92	15888.33	-
SVR + Linear	0.9056	8276.25	11885.83	68
Artificial Neural Network	0.7521	12149.54	17966	29
Random Forest	0.8382	9336.13	14825.88	42

Table 2: Average Validation Scores: Accuracy, MSE, MAE across best models

Models	**Test Accuracy - R2 score**	**RMSE**	**MAE**
Trivial Method	-0.00499	27918.49	35943.85
Baseline (Linear)	0.6331	15975.83	21715.72
SVR + Linear	0.7469	11560.60	18037.96
Artificial Neural Network	0.7521	12149.54	13796
Random forest	0.7442	11631.41	18133.70

Table 3: Test Scores: Accuracy, MSE, MAE across best models

5 Libraries used and what you coded yourself

The following table summarizes the Python libraries that were used as is and the ones that were coded by the team.

Libraries Used	**Functions/Modules Self-Coded**
numpy	Linear Regression Model
math	Polynomial Regression
pandas	KNN
seaborn	ANN
torch	R2 score
matplotlib.pyplot	MAE
sklearn	MSE
SequentialFeatureSelector	
re	
mlxtend	

Table 4: Summary of Libraries Used and Functions Coded

6 Contributions of each team member

Task	Khushboo	Neha
Dataset Understanding	50%	50%
EDA	50%	50%
Pre-Processing and Feature Engineering	80%	20%
Model Implementations	20%	80%
Evaluation and Testing	50%	50%
Report	50%	50%

Table 5: Task distribution Between Team Members.

7 Summary and conclusions

In this project, we aimed to optimize the preprocessing procedures on a dataset of laptop specifications and subsequently assess its performance across various baseline and advanced machine learning regression models to identify the most effective predictive model. We employed a range of feature selection strategies as a part of our feature engineering efforts to enhance the dataset's representation, including the implementation of univariate feature selection, backward sequential feature selection and one-hot encoding. The evaluation of model efficacy was conducted using key statistical metrics, namely R^2 score, Root Mean Square Error (RMSE), and Mean Absolute Error (MAE). The analysis demonstrated that models such as Support Vector Regression (SVR) with a linear kernel, Random Forest, and Artificial Neural Network exhibited superior performance, which were subsequently applied to the test dataset.

In the future, we intend to further refine our approach by exploring more sophisticated preprocessing techniques and experimenting with model stacking to devise a unique model architecture that might more effectively capture the intricacies of the dataset features and give superior performance.

References

[1] Alice Zheng and Amanda Casari, *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*, O'Reilly Media, Inc., 2018.

[2] Vitor Werner de Vargas, Jorge Arthur Schneider Aranda, Ricardo dos Santos Costa, Paulo Ricardo da Silva Pereira, Jorge Luis Victória Barbosa, *Imbalanced data preprocessing techniques for machine learning: a systematic mapping study*, *Knowledge and Information Systems*, vol. 65, no. 1, pp. 31–57, Springer, 2023.

[3] Steve R. Gunn et al., *Support vector machines for classification and regression*, ISIS technical report, vol. 14, no. 1, pp. 5–16, 1998.

[4] Richard G Brereton and Gavin R Lloyd, *Support Vector Machines for Classification and Regression*, *Analyst*, vol. 135, no. 2, pp. 230–267, 2010, Royal Society of Chemistry.

[5] Victor Rodriguez-Galiano, Manuel Sanchez-Castillo, M Chica-Olmo, MJOGR Chica-Rivas, *Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines*, in Ore Geology Reviews, vol. 71, pp. 804–818, Elsevier, 2015.

[6] Victor Rodriguez-Galiano, Manuel Sanchez-Castillo, M. Chica-Olmo, and MJOGR Chica-Rivas, *Machine learning predictive models for mineral prospectivity: An evaluation of neural networks, random forest, regression trees and support vector machines*, *Ore Geology Reviews*, vol. 71, pp. 804–818, 2015, Elsevier.

[7] https://towardsdatascience.com/how-to-build-a-bayesian-ridge-regression-model-with-full-hyperparameter-integration-f4ac2bdaf329