Laptop Price Prediction

EE 559 Course Project

Data Set: Laptop Prices

Author, Bhavya Samhitha Mallineni USCid: mallinen@usc.edu

Apr 26, 2024

1. Abstract

This project aims to predict laptop prices using a dataset of 1,300 entries, employing machine learning models to handle various laptop configurations. We developed three models: Linear Regression, Support Vector Regression (SVR), and a Neural Network, each addressing different dataset complexities. Linear Regression was applied using the Moore-Penrose Pseudoinverse and a gradient descent method, SVR utilized an RBF kernel to manage non-linear relationships, and the Neural Network used a single-layer architecture to learn data patterns. Evaluation metrics, including Root Mean Square Error (RMSE), Mean Absolute Error (MAE), and R-squared (R^2), assessed the models' performances.
The implementation leveraged Python libraries such as pandas, numpy, and sklearn for data preprocessing, model construction, and performance evaluation. Notably, SVR emerged as the superior model, excelling in all evaluation metrics, indicating its effectiveness in capturing complex patterns in the data and providing accurate price predictions. This study underscores the utility of advanced regression techniques in predictive analytics, offering significant insights for technological market applications.

2. Introduction

2.1 Problem Assessment and Goals

The laptop dataset comprises 11 columns detailing the features of each laptop along with its corresponding price. Our objective is to maximize the utility of these features for accurate price prediction when presented with various laptop configurations. Given the multifaceted nature of laptop pricing, we employ regression models to forecast these prices. Through this approach, we aim to leverage the dataset's features effectively, thereby enhancing the precision of our price predictions.

2.2 Literature Review

For preprocessing the data in the laptop dataset, I referred to the code section on Kaggle [1] to get some ideas. The code section on Kaggle offered various approaches for converting categorical data into numerical form. Some methods involved label encoding, while others utilized one-hot encoding. Drawing inspiration from these approaches, I devised my own preprocessing method.Several Sklearn articles were also referred to use the libraries in the code

3. Approach and Implementation

3.1 Dataset Usage

The Laptop dataset consists of 1300 data points and 11 columns. For training purposes, we've split the dataset into a training set of 900 data points and a test set of 400 data points. Among these columns, one column represents the target value, which is the price, while the remaining 10 columns are features. Of these 10 features, 9 are categorical, and 1 is numerical. We've employed 5-fold cross-validation, which involves running the model 5 times. Each time, the data is split into 4 training sets (720 data points each) and 1 validation set (180 data points).

The features in the dataset are

1. Company - categorical
2. TypeName - categorical
3. Inches - numerical
4. ScreenResolution - categorical
5. Cpu - categorical
6. Ram - categorical
7. Memory - categorical
8. Gpu - categorical
9. OpSys - categorical
10. Weight - categorical

3.2 PreProcessing

The primary objective of my preprocessing was to convert the mostly categorical data in the training set into a numerical format suitable for training models. I started by checking the feature space for missing or null values, but none were found. The only numerical feature, 'Inches', required normalization as it was not standardized. Normalization is crucial to ensure that features with different scales contribute equally to the model training process, preventing features with larger magnitudes from dominating those with smaller magnitudes.

The first step was 'Data Cleaning', where I separated the 'gb' and 'kg' units from the 'Ram' and 'Weight' features, respectively. After removing these units, I normalized the values to ensure they ranged between 0 and 1. Next, I examined the 'Company', 'TypeName', and 'OpSys' features to determine the unique categories and their counts, assessing their importance. These three features were then one-hot encoded. One-hot encoding is necessary to convert categorical variables into a numerical format that machine learning algorithms can interpret. It creates binary columns for each category, eliminating any ordinal relationship between categories and allowing the model to understand the categorical data properly.

3.3 Feature Engineering

The dataset included features like 'ScreenResolution', which contained multiple attributes within a single feature, such as 'IPS Panel Retina Display 2560x1600'. These attributes were parsed into six separate features, including 'FullHD', 'IPS', 'TouchScreen', 'Retina', 'ScreenWidth', and 'ScreenHeight'. After this segmentation, the new features underwent normalization.

Feature engineering is important because it allows us to extract meaningful information from raw data and create new features that better represent the underlying patterns in the data. In this case, parsing 'ScreenResolution' into separate features allows us to capture specific characteristics of the screen technology, which can be important for predicting laptop prices.

Similar transformations were applied to the 'Cpu' feature, which was divided into 'Cpu Speed' and 'CPU Company', and to the 'Memory' feature, where values were converted to gigabytes (GB) for consistency and then split into 'Storage Size' and 'Storage Type'.

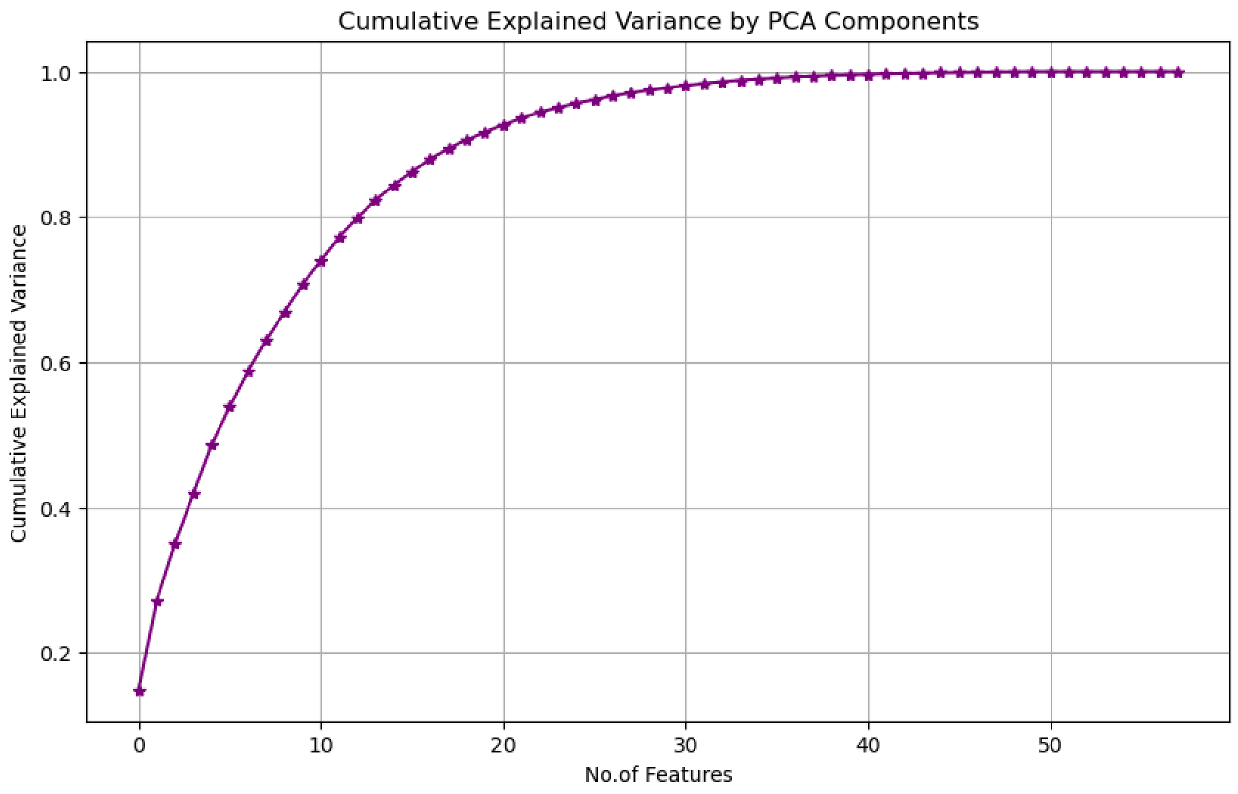
However, the 'GB' feature contained a wide range of values (over 100 types), making it impractical to break down into a few simple features. To address this, we employed rare label encoding, where the most frequent categories were separated into new features while infrequent ones were labeled as miscellaneous. Rare label encoding is used to handle categorical variables with a large number of unique values, ensuring that the model does not overfit to rare categories and focusing on the most common and informative ones.

3.4 Feature Dimensionality Adjustment

Following preprocessing, differences emerged between the number of features in the training and test sets due to potential missing categories in the smaller test set. To address this, I identified features present in the training set but missing in the test set and added them to the test set with zero values. While an alternative approach would involve removing these features from the training set, I opted for the former method to maintain consistency in feature dimensions across both sets.

Subsequently, the number of features expanded significantly, from the original 10 to 57. However, not all of these features are essential for training, and many may have minimal impact. To address this issue of high dimensionality and reduce computational complexity, I employed Principal Component Analysis (PCA). PCA is a technique used for dimensionality reduction that identifies the most informative components in the data while minimizing information loss. By retaining components that explain a significant portion of the variance, PCA enables the reduction of feature dimensionality while preserving the essential information needed for modeling.

PCA involves transforming the original features into a new set of orthogonal components, known as principal components, which capture the maximum variance in the data. These components are computed using linear algebra techniques such as singular value decomposition or eigendecomposition. The importance of PCA lies in its ability to simplify complex datasets by representing them in a lower-dimensional space while retaining as much relevant information as possible. This not only reduces computational overhead but also helps mitigate issues such as overfitting and the curse of dimensionality, ultimately improving the efficiency and effectiveness of machine learning models. After PCA the first 37 features were taken into account. After applying PCA, I selected the top 37 features based on their contribution to explaining the variance in the data. This selection process ensures that we retain the most significant information from the original dataset while reducing its dimensionality.



.

3.5 Regression Model Training

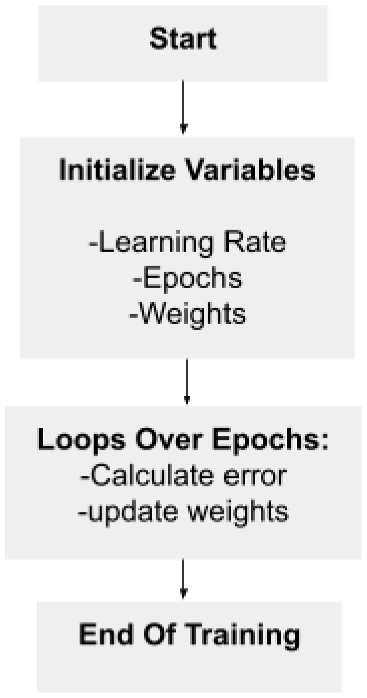
3 Models where used to Predict the Price

1. Linear Regression:

Linear regression is a technique used to establish a relationship between the dependent variable and the independent variable, enabling us to predict the dependent variable based on the independent variable. In our scenario, we aim to predict the price using features. We employed two methods to implement the linear regression model. The first method involves utilizing the Moore-Penrose Pseudoinverse in class to determine the optimal weights. The second method employs the mean squared error (MSE) algorithm and gradient descent, similar to one of the homework problems. With this approach, we continually adjust the weights until we achieve a very low MSE value between the actual target and the predicted target variables.

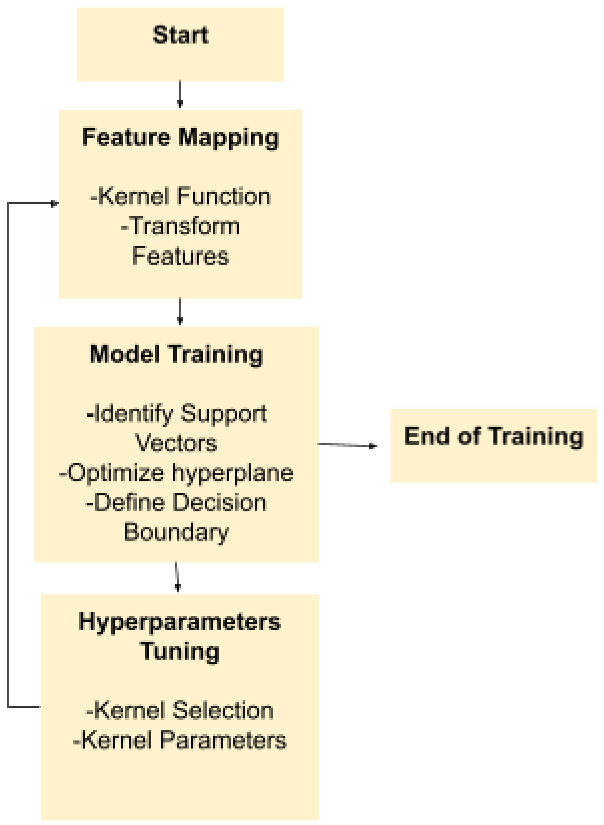
For the gradient descent, the learning rate was taken as 0.01 and the number of epochs as 1000. The parameters were chosen based on the low error achieved with different trials. The degrees of freedom (DOF) for this model is (D+1), which equals 38 (37 features plus 1 intercept). The number of constraints is 720 for k-fold validation and 900 for normal training.
Moore-Penrose Pseudoinverse : w = (X^T X)^-1 X^T y
MSE:- y_hat = X * w + b
Error =y_true - y_hat
Gradient: w = w - lr * dJ/dw

Algorithm For Linear Regression:



2. Support Vector Regression

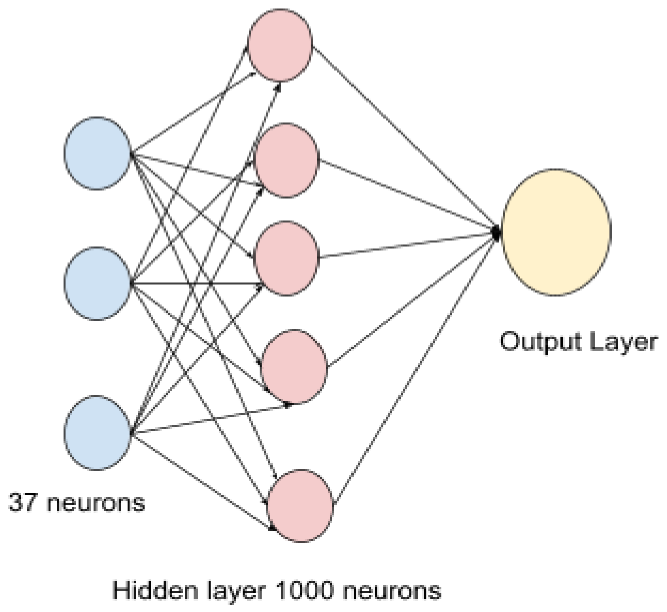
For the next model I have built a Support vector regression model with an rbf kernel. SVR constructs a hyperplane in a high-dimensional space that best fits the training data while maximizing the margin between the hyperplane and the data points. It employs a kernel function to map input features into a higher-dimensional space, enabling it to capture complex relationships between features and the target variable. During training, SVR identifies a subset of data points called support vectors and optimizes a hyperplane to minimize the error between the predicted and actual target values. SVR has hyperparameters such as the choice of kernel function, regularization parameter (C), and kernel parameters, which need to be tuned for optimal performance. After training, SVR can make predictions on new data points by applying the learned hyperplane. An RBF kernel is a gaussian kernel, and is used for getting non-linear relationships between input features. Gamma value here affects the smoothness of the decision boundary. For our model, rbf kernel with C = 10, and gamma= 0.1. Different parameters were tested, and the one resulting in the lowest mean RMSE across k-fold cross-validation was selected. This model was constructed using the SVR function from the sklearn library.



3. 1 Neural Network

For the third model I have taken a neural network of 1 layer. A neural network works by learning patterns and relationships in data to make predictions or classifications. It consists of interconnected layers of neurons that process input data, extract features, and produce output predictions. During training, the network adjusts its parameters (weights and biases) using optimization algorithms like gradient descent to minimize prediction errors. This process

involves forward and backward passes: in the forward pass, input data is passed through the network to generate predictions, while in the backward pass, prediction errors are propagated back through the network to update parameters. Through iterative training on labeled data, the network learns to generalize and make accurate predictions on unseen data. The network's architecture, activation functions, and optimization methods play crucial roles in its performance and ability to learn complex relationships in the data.
For my model, I performed k-fold validation and tested various hyperparameters. The configuration that achieved the best performance included 1000 neurons in the hidden layer, the Adam optimizer with a learning rate of 0.01, and a regularization factor of 0.0001. This model was built using the MLPRegressor class from the sk-learn library.The DOF for this model is (37 x 1000) +1000 = 38000.



4. Results and Analysis

Comparison between Trivial System and 3 Baseline System.

Trivial System:
Taking the predicted target value as mean of all the target training points
We get

RMSE Trivial System	0.63

MAE Trivial System	0.52
R2 Score Trivial System	-0.01

Which is pretty high and can infer that our model using a trivial system would not be making good predictions.

Baseline System:

Linear Regression:
Built using the sklearn linear regression model.

RMSE Baseline Linear Regression	0.30
MAE Baseline Linear Regression	0.23
R2 Score Baseline Linear Regression	0.77

1NN:
Built using sklearn MLPRegressor

RMSE Baseline Linear Regression	0.39
MAE Baseline Linear Regression	0.30
R2 Score Baseline Linear Regression	0.61

From these initial values we can see that changing the hyperparamters using cross validation has improved the prediction.

3 Models

1. **Linear Regression:**

a) **Pseudo Inverse Solution:**

Cross Validation:

Mean RMSE on Validation Set	0.26
Standard Deviation on Validation Set	0.02
MAE on ValidationSet	0.21

R2 Score	0.81

Test Set

RMSE	0.30
MAE	0.23
R2 Score	0.77

b) Gradient Descent MSE Algorithm

Test Set

RMSE	0.34
MAE	0.27
R2 Score	0.71

Analysis and Observations:

The linear regression model was initially trained using the pseudoinverse method, resulting in a relatively low error rate for the validation set. When evaluated on the test set, the error rate was slightly higher but still acceptable. Since the pseudoinverse method does not involve hyperparameters, an alternative linear regression approach was explored. Various learning rates were tested with different numbers of epochs, with 0.01 yielding the lowest error rate among them. Thus, it was selected for the final model. However, despite optimization attempts using gradient descent to obtain optimal weights, the error rate remained higher compared to the pseudoinverse method.

2. Support Vector Regression:

Cross Validation:

Mean RMSE on Validation Set	0.23
Standard Deviation on Validation Set	0.01

MAE on ValidationSet	0.18
R2 Score	0.86

Test Set

RMSE	0.25
MAE	0.20
R2 Score	0.84

Analysis and Observations:

For SVR, three parameters were investigated to minimize error: Kernel type, C, and gamma values. Initially, the linear kernel yielded validation set errors around 0.3. Upon trying the RBF kernel with default sklearn library C and gamma values, the mean RMSE for the validation set was 0.28. Consequently, RBF was chosen as the kernel, and further optimization was conducted to find the best C and gamma values. When C ≤ 1 and gamma ≥ 0.01, all resulting RMSE values exceeded 0.4. However, with 1 < C < 10, the error decreased to around 0.3. The lowest error of 0.23 was attained with C = 10 and Gamma = 0.1, making them the final hyperparameters. Testing was subsequently conducted using these parameters.

3. **Neural Network:**

Cross Validation:

Mean RMSE on Validation Set	0.28
Standard Deviation on Validation Set	0.04
MAE on ValidationSet	0.21
R2 Score	0.79

Test Set

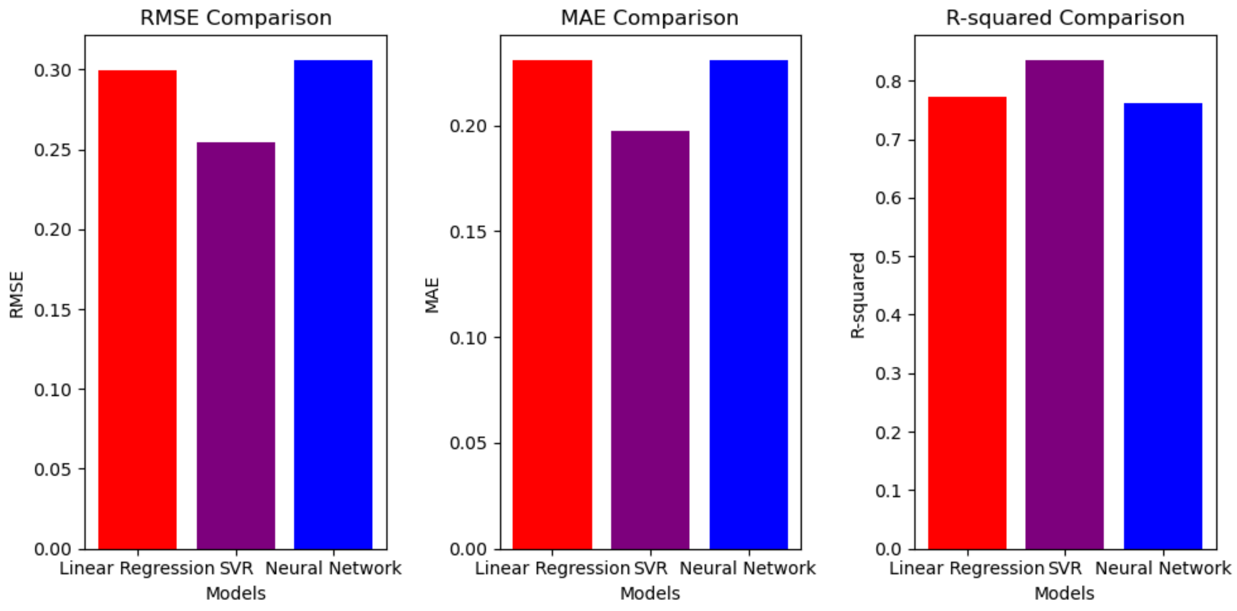
RMSE	0.31
MAE	0.23

R2 Score	0.76

Analysis and Observations:

Similar to the previous models, different hyperparameters were experimented with to minimize error in the neural network. It was observed that increasing the complexity of the neural network tended to reduce error rates. However, with only one hidden layer, the error was not significantly reduced. Notably, increasing the number of neurons in the hidden layer from 100 to 1000 resulted in improved performance. Beyond this point, further increasing the number of neurons did not yield better results. The optimal hyperparameters identified were: learning rate = 0.01, regularization coefficient = 0.0001, optimizer = Adam, and 1000 neurons in the hidden layer. These parameters were selected based on their ability to minimize error during training and validation.

The models improves well from its trivial and baseline stages when hypertuning was performed. Here is a comparison between the 3 models that we used.



5. Libraries Use and Code Implementation

For this project, several libraries were utilized, including pandas, numpy, and sklearn. Specifically, the kfold, SVR, and MLPRegressor libraries were employed for modeling. Additionally, preprocessing tasks were conducted using modules such as Label Encoder, MinMaxScaler, and PCA from the sklearn preprocessing module. Metrics from the sklearn.metrics library were utilized to calculate key evaluation metrics, including RMSE, MAE, and R2. Notably, the linear regression model was implemented from scratch, demonstrating a hands-on approach to constructing the model. This comprehensive utilization of libraries and custom implementation allowed for efficient preprocessing, modeling, and evaluation of the predictive models in the project.

6. Summary and Conclusions

In conclusion, based on the evaluation metrics of RMSE, MAE, and R2 score, the SVR model with tRBF kernel, C=10, and gamma=0.1 outperforms the linear regression models and neural network. It achieves the lowest RMSE and MAE values on both the validation and test sets, indicating superior predictive accuracy. Additionally, its higher R2 score suggests better overall fit to the data. Therefore, the SVR model is the most suitable choice among the three models evaluated, demonstrating its effectiveness in capturing the underlying patterns in the data and making accurate predictions.

For future, we can implement a deeper neural network with multiple hidden layers which can help in decreasing the errors and giving a better prediction.

References

[1]https://www.kaggle.com/datasets/ara001/laptop-prices-based-on-its-specifications/code only reference for a general idea ws taken, no concrete code was used from this.

[2] https://scikit-learn.org/stable/modules/model_evaluation.html

[3] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html