Laptop Price Prediction With Machine Learning Approaches

1. **Dataset Overview:**

   This project uses the laptop prices dataset provided, and was originally sourced from Kaggle. The dataset contains 1,303 laptop entries with structured specification information and a continuous target variable representing laptop price in euros.

   The features describe both hardware and physical characteristics of each laptop, including memory size, CPU and GPU information, screen properties, weight, product type, and manufacturer. Several features are categorical in nature and require encoding before modeling.

2. **Project Goal Overview:**

   The goal of this project is to predict laptop prices using structured specification data. Given hardware and product features such as memory, processor, graphics, display properties, and manufacturer, we model laptop price as a continuous regression target. The objective is to evaluate and compare different regression based machine learning approaches to determine which methods most effectively capture the relationship between laptop specifications and price.

3. **Methodology that Worked & Explanation**

   To start with, data preprocessing and EDA were critical to achieving good model performance. Numeric features were scaled to ensure comparable magnitudes, while categorical variables were one-hot encoded. The target variable was log transformed to reduce right skew and stabilize variance. Exploratory data analysis showed that laptop price is strongly influenced by hardware specifications such as RAM, CPU/GPU, and screen resolution, while brand related features introduce additional variability and nonlinear effects. These observations motivated the use of regularized and nonlinear models that can handle these types of multicollinearity and feature interactions.

   Comparing final model performance, ridge regression, XGBoost, and the neural network appeared to be equally strong with very similar test performance. Ridge regression achieved a test R^2 of 0.8881 with RMSE 0.0398 and MAE 0.0275, demonstrating that L2 regularization effectively stabilizes linear models in high dimensional settings. XGBoost achieved the best overall performance with a test R^2 of 0.8963, RMSE 0.0383, and MAE 0.0275 due to its ability to model nonlinear relationships and feature interactions. The tuned neural network with early stopping achieved comparable results as well, with a test R^2 of 0.8862, RMSE 0.0401, and MAE 0.0274, indicating neural networks can perform competitively when properly regularized and tuned, even with limited dataset and high-dimensional input space.

4. **Key Method & Limitations**

   Model performance was evaluated by MSE, RMSE, MAE, and R^2. A consistent train test split was used across all models to enable direct comparison.

   XGBoost was selected as the key method based on its overall performance. It achieved the highest test R^2 of 0.8963 and the lowest RMSE of 0.0383 among the evaluated models, with MAE comparable to the best performing alternatives. Although Ridge regression and the neural network showed similar results, XGBoost consistently provided the strongest performance across metrics, making it the most reliable choice for this task.

Despite its strong performance, XGBoost has limitations. The model is less interpretable than linear regression, which makes it harder to directly understand feature contributions. In addition, its performance can be sensitive to hyperparameter choices, and further tuning may improve results. Exploring interpretability techniques and additional model refinement would be valuable extensions of this work.

5. How to use the code

Users should first upload the 4 preprocessed datasets generated during preprocessing, X_train_scaled.csv, X_test_scaled.csv, y_train.csv, and y_test.csv before model training and evaluation. The full project pipeline, including preprocessing, linear models, PCA & Clustering, XGBoost, and neural network, is contained in the main notebook. Running the notebooks entirely will reproduce all reported results using the same train test splits and evaluation metrics.

Appendix

i. Explain the exploratory data analysis that you conducted. What was done to visualize your data and split your data for training and testing?

We first examined the distribution of Price_euros by obtaining descriptive statistics for the training set's response variable (mean, median, variance, IQR, etc.). The distribution was visualized using a histogram, which showed a heavy right skew. This led us to apply a log-transformation to normalize the data for linear regression. Correlation was examined using a heat map of the 10 features most correlated with Price_euros. Notably, there was a strong correlation between ScreenH and ScreenW, and between Inches and Weight, suggesting potential multicollinearity. Extracting the correlations for these features confirmed this, further motivating the use of regularization in our linear regression model. The dataset was split into 80% training and 20% testing sets, with a fixed random_state=42 to ensure reproducibility.

ii. What data pre-processing and feature engineering (or data augmentation) did you complete on your project?

After inspecting the data, we found that there were no missing values or values that needed to be imputed. One-hot encoding was used to transform categorical data (e.g. Company, TypeName, GPU_model) into numerical values. We applied min-max scaling to the numerical features (e.g., Inches, Ram, Weight) to normalize them to values between 0 and 1. To prevent data leakage, the scaler was fitted on the training set only, then used to transform the test set.

iii. How was regression analysis applied in your project? What did you learn about your data set from this analysis and were you able to use this analysis for feature importance? Was regularization needed?

We applied three regressions: linear regression, linear regression with ridge regularization, and linear regression with lasso regularization. Based on the regression results, we learned that the dataset is high-dimensional and contains many weak or redundant predictors. Regularization is necessary here because regression without regularization produces a negative R^2 value, indicating tremendous overfitting. Regularization greatly improved generalization, showing that the underlying feature relationships are noisy and highly correlated. Lasso further revealed that only 26 features carry meaningful signals, while 848 contributed almost nothing. Both ridge and lasso regularization successfully fixed this issue.

iv. How was logistic regression analysis applied in your project? What did you learn about your data set from this analysis and were you able to use this analysis for feature importance? Was regularization needed?

Logistic regression analysis was applied as a part of the EDA for our problem. We binarized the Price_euros variable as either high-end (greater than or equal to the median) or budget (less than the median), and used this to assess the relevance of various features. By default, LogisticRegression applies ridge regularization (L2), and this was necessary for the most accurate predictions. We learned that the price range of a laptop could be predicted with decent accuracy, with a high rate of true positives. Because our data was standardized, the regularized coefficients could be directly examined for feature importance. Based on the coefficients after ridge and lasso regression, RAM, CPU/GPU, and screen resolution seem to be important features, with specific brand-related features introducing additional variability.

v. How were KNN, decision trees, or random forest used for classification on your data? What method worked best for your data and why was it good for the problem you were addressing?

We evaluated KNN across a range of neighbors k from 1 to 20, using 5 fold cross-validation with MAE as the performance metric to find the best value for k. Since having a lower MAE is better, we used sklearn's neg_mean_absolute_error and converted the result back to positive MAE. The best k value, indicated by the minimum MAE, was 4. Using k=4, we trained the KNN model and used R^2, MSE, and MAE to evaluate its performance.

- Test R^2 = 0.8481
- Test Mean Squared Error (MSE) = 0.0021
- Test Mean Absolute Error (MAE) = 0.0329

For random forest, we built 100 trees and combined their predictions by averaging. Each decision tree was trained on a different random subset of the training set. That reduced the influence of outliers and decreased model variance. We used R^2, MSE, RMSE, and MAE to evaluate its performance.

- Test R^2 = 0.86922
- Test Mean Squared Error (MSE) = 0.00185
- Test Root Mean Squared Error (RMSE) = 0.04300
- Test Mean Absolute Error (MAE) = 0.02906

From the data, random forest is better than KNN here. Random forests worked best for the data. Individual decision trees tend to overfit, but random forest combines many trees and average their prediction to reduce overfitting. Whereas for KNN, it highly relies on distance metrics that require feature scaling to perform well, while random forest does not depend on it. Overall, random forest leads to the highest predictive accuracy on the dataset.

vi. How were PCA and clustering applied on your data? What method worked best for your data and why was it good for the problem you were addressing?

Applying PCA to the eight numeric features worked well and suggested that the first three to four principal components captured most of the variance. Using the first two PCs, K-means clustering with k=4 produced loosely separated clusters in PCA space. The clusters were somewhat distinct but sparsely distributed, suggesting that, while numeric features have some structure, clustering is not very useful for predicting laptop price. On the other hand, PCA is useful for the problem we are addressing here since it reduces hundreds of features to about four PCs, greatly cutting down on dimensionality and helping us understand the underlying structure.

vii. Explain how your project attempted to use a neural network on the data and the results of that attempt.

We trained a simple feedforward neural network on the scaled feature set using two hidden layers. The model learned the training data well and reached a test R^2 of around 0.8, showing that it was able to capture some nonlinear relationships in the dataset. However, its performance was still worse than Ridge regression. The high-dimensional sparse input space and limited data size make regularized linear models more effective for this task than NN.

viii. Give examples of hyperparameter tuning that you applied in preparing your project and how you chose the best parameters for models.

We applied hyperparameter tuning for KNN to improve predictive performance. To select the best k, we evaluate 1 to 20 using 5 fold cross-validation with MAE as performance metric. Since sklearn reports negative MAE, we use neg_mean_absolute_error and convert the result back to positive MAE. The value of k that minimizes the MAE was 4. So we select k=4 as the optimal hyperparameter that produces the best generalization performance on the validation data.