Main Document

Dataset Used

We used the Laptop Prices dataset, which contains information about different laptops and their specifications, such as price, brand, type, screen size, operating system, etc. It contains 1023 laptops and 23 features.

Problem Statement

We wanted to classify a laptop's type (i.e. whether a laptop is a gaming laptop, 2-in-1 convertible, notebook, etc) given its specifications (amount of RAM, screen size, company, etc).

Methodology

We decided to use a random forest model for our multi-class classification of laptop type. It handles both numeric and categorical features well, and the dataset contains a mixture of both. It can also capture non-linear relationships and handle outliers due to it being an ensemble method that takes the output of multiple models into account. It also performed better than the other models we trained.

To further increase accuracy, we performed hyperparameter tuning. We compared different results after adjusting the number of trees, the maximum depth on each tree, the minimum number of samples required to split an internal node, and the minimum number of samples to be in a leaf node. To select the best combination of hyperparameters, we used grid search to identify the best value for each hyperparameter. Doing so evaluates every combination of the hyperparameter values, guaranteeing that we find the optimal model within the values we chose to search. After comparing results, the best-performing model had 32 trees, a maximum depth of 10, and a minimum sample split of 3. We also used a 5-fold cross-validation strategy, splitting the dataset into four training sets and one validation set for each iteration. This way, the model would be tested on different subsets to reduce overfitting.

Results

```
Test Accuracy: 0.8078

Classification Report on Testing Set:
                    precision    recall  f1-score   support

2 in 1 Convertible       0.64      0.39      0.49        23
            Gaming       0.76      0.83      0.79        35
           Netbook       0.00      0.00      0.00         1
          Notebook       0.86      0.92      0.89       144
         Ultrabook       0.77      0.72      0.75        47
       Workstation       1.00      0.20      0.33         5

          accuracy                           0.81       255
         macro avg       0.67      0.51      0.54       255
      weighted avg       0.81      0.81      0.80       255
```

We achieved an overall test accuracy of 80.8%, meaning the model classified approximately 81% of the samples correctly. The majority class (Notebook) achieved the highest precision and recall score because it had the most examples that the model could train on. The classes with moderate support also achieved a decent recall. However, the classes with fewer than 10 examples (workstations and netbooks) perform poorly because of the lack of training data. As a result of the class imbalance, the macro average is lower (0.54) but the weighted average is relatively high (0.8).

Appendix

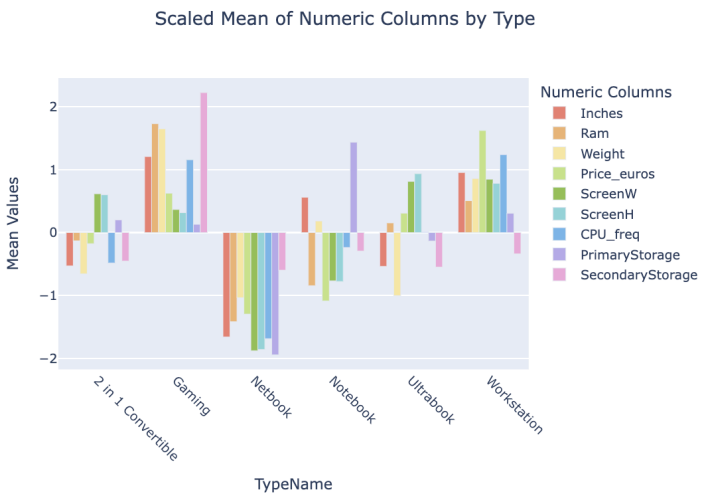
Exploratory Data Analysis

To begin, the data was loaded into the notebook and inspected for errors after loading. We parsed it as a CSV and checked that the column names and dimensions matched the original dataset's description. Both matched what was described in the source dataset's Data Card (1275 datapoints, 23 features). Upon inspecting the data for duplicated samples and NA/null values, we found none.

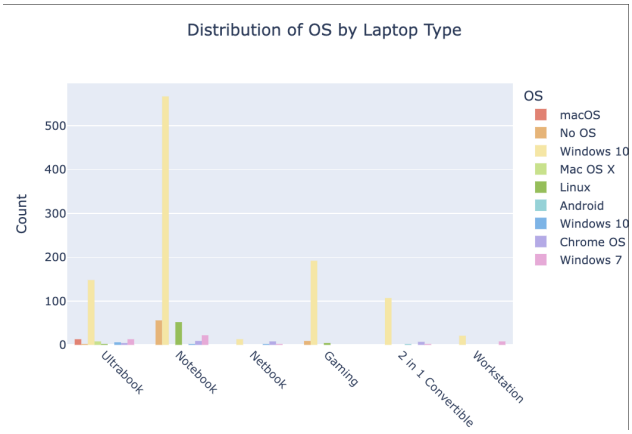
Visualization of Data

Initially, we conducted a basic descriptive analysis by grouping the data by laptop type and calculating the average values for every numerical feature. For example, gaming

laptops are £1731.38 and 2.95 kg on average. Before we visualized this data, we standardized the features so that we could view them on the same scale by using z-score normalization. By making sure that every feature was centered with a mean of 0 and a standard deviation of 1, the visualization could be easier to interpret. We then plot these values on a bar chart to find patterns between different types of laptops and their features. This gave us a general idea of patterns within different types of laptops' features, but further analysis was needed to see which features most strongly correlated with type.



The previous visualization explored relationships between numerical values and laptop type. Next, we used a chi-square test of independence to see if there were any significant relationships between each categorical variable and the type of computer. The outputted p-values are extremely small (from 2.3625361706290534e-133 for GPU_company to 2.0965450168795287e-05 for CPU_company), indicating that the categorical features are very closely related to the type of computer. This confirms that the categorical variables (company, OS, screen type, etc.) have a large contribution to the classification of laptop type. Histograms were created to visualize these relationships.



Train and Test Splits

We created two different data splits. For methods that didn't require a validation dataset, we randomly split the dataset into a training and validation set. 80% of the dataset is used for training and 20% for testing. This makes sure that the data is not being overfitted. We used this to train our first few models: a linear regression model and a binary logistic regression model. Because we didn't have hyperparameters to tune for these models and they were not a good fit for our use case, we decided a validation set was not necessary.

Our second split was used to train our neural network. We split the dataset into training, validation, and test sets at 70%, 20%, and 10% of the original dataset each. We used the validation set to select models and values for hyperparameters such as learning rates, dropout rates, epochs, and batch sizes.

Data Preprocessing and Augmentation

Our initial data preprocessing involved inspecting the dataset for null/NaN and duplicated values. For both, we found zero occurrences. The dataset linked on Bruinlearn contained some preprocessing code that was already executed on the dataset. It cleaned and prepared the original data while transforming some features. The ScreenResolution feature was originally of the format "1000x2000", but the dataset authors extracted the numerical values and created two new columns - width and height. Similarly, the CPU column was transformed and split into the CPU_company,

CPU_model, and CPU_freq columns. In addition to extracting numerical values, the preprocessing code also removes units (i.e. 8 instead of “8GB”, 1.5 instead of “1.5kg”) and drops any duplicate samples. This transforms various categorical features that would make more sense as numerical values, making it easier for machine learning models to parse.

We also performed z-score normalization on the numerical columns and performed one-hot encoding on the categorical data. Furthermore, we encoded the TypeName (laptop type - Notebook, gaming laptop, etc.) into a numerical value under a column called “‘LabelEncodedType’”. The one-hot encoding transforms the categorical data into a format that machine learning models can interpret, and the z-score normalization ensures that the scaling across features is similar.

Regression Analysis

Linear Regression

Despite one-hot encoding the type of laptop into a numerical value, linear regression is not a good fit for classifying laptop types. It's designed for regression tasks, and laptop type is not a continuous numeric value. Because we encoded TypeName into numeric values (i.e. 3 is assigned to Notebook, 4 is assigned to Ultrabook), there’s an assumed ordinal relationship between types (Notebook < Ultrabook) when this assumption isn’t exactly sound. This mismatch between task and model explains the poor error metrics:

Training Metrics: {'Mean Absolute Error': 0.81, 'Mean Squared Error': 1.32, 'Root Mean Squared Error': 1.15, 'R2 Score': 0.16}

Applying ridge regularization did not have much effect on the output, and the values were almost exactly the same. We couldn’t derive much meaningful analysis from this model’s results, but it helped us decide on using classification-based approaches instead.

Logistic Regression

We used logistic regression to perform binary classification of laptops based on price categories: samples could either be classified as high-priced if they're a Workstation, a Gaming laptop, or an Ultrabook or as low-priced if they're a 2 in 1 convertible, notebook, or netbook. Though we had to semi-arbitrarily sort laptop types into two categories to make our statement suit the purpose of a logistic regression model better, the accuracy was still higher than linear regression. We applied L2 regularization, but it yielded the same results as the model without regularization. Our training accuracy is not too different from our testing accuracy, indicating that the models might not be overfitting and thus regularization is not necessary.

```
Logistic Regression without Regularization
Accuracy: 0.8235

Classification Report without Regularization:
              precision    recall  f1-score   support

           0       0.82      0.94      0.88       168
           1       0.84      0.60      0.70        87

    accuracy                           0.82       255
   macro avg       0.83      0.77      0.79       255
weighted avg       0.83      0.82      0.81       255
```

The majority class (Class 0: low-cost laptops) has a much higher recall (0.94) than the minority class (Class 1: high-cost laptops), indicating that high-cost laptops were harder to classify. This might be due to the class imbalance or the way we chose to classify laptops as higher or lower cost.

KNN, Decision Tree, Random Forest Classification

In our classification task, we tested three models: K-Nearest Neighbors (KNN), Decision Trees, and Random Forest, comparing their test accuracies and overall performance. Among these, Random Forest emerged as the best-performing model, achieving the highest test accuracy of 80.8%. Because it combines multiple decision trees to reduce overfitting, handle both numerical and categorical features, and capture complex non-linear relationships in the data, it outperformed the other methods. These

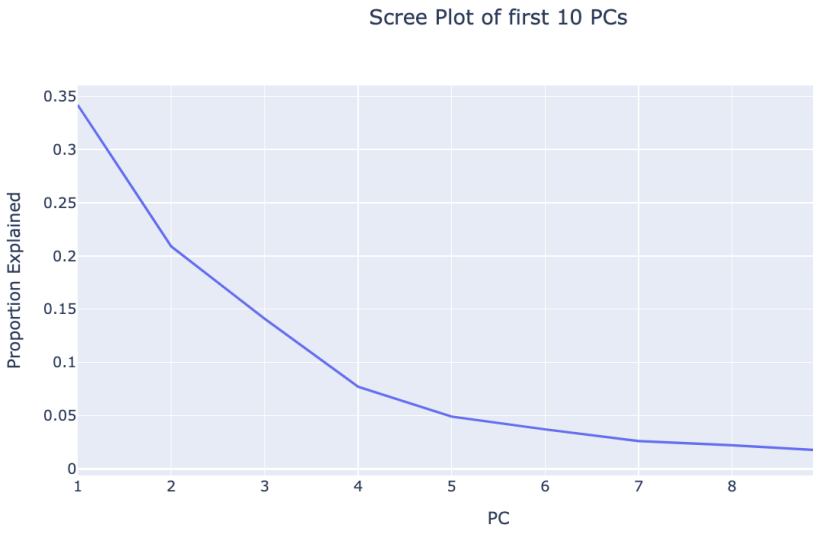
characteristics make Random Forest particularly suited for multi-class classification tasks like ours.

KNN achieved a test accuracy of 77%, demonstrating its simplicity and effectiveness. However, it doesn't perform as well on our dataset with many features. It also lacks the scalability and robustness offered by the ensemble-based Random Forest.

The Decision Tree model performed the worst, with a test accuracy of 67%. This is likely due to overfitting. While Decision Trees are interpretable and computationally efficient, their limited ability to generalize well to unseen data makes them less effective for complex classification problems.

Principle Component Analysis

We used Principal Component Analysis to transform the dataset into a lower-dimensional space based on the explained variability of each principal component. To do this, we created a Scree Plot to visualize the elbow point at which point adding more PCs no longer gives significant returns. Based on the plot, 5 PCs were chosen, which explain 85.45% of the cumulative variability.



We used Principal Component Analysis to transform the dataset into a lower-dimensional space based on the explained variability of each principal component. To do this, we created a Scree Plot to visualize the elbow point to decide

how many PCs to choose. After this point, adding more PCs no longer gives significant returns in capturing variability. Based on the plot, 5 PCs were chosen, which explains 85.45% of the cumulative variability.

Because the original data had a relatively higher number of features, the dimensionality reduction provided by PCA is helpful in reducing the number of features that don't greatly impact the variance in the data. This makes training models on this data less computationally expensive. It also reduces collinearity by transforming the original dataset's features into uncorrelated principal components and reduces the chance of overfitting by reducing the complexity of the model and dataset.

Neural Networks

We decided to use a simple feedforward neural network. We chose a simple FNN because of our dataset's structure - it's tabular and doesn't have many features or a large number of samples. We also reduced a lot of collinearity through PCA, so each feature should more independently contribute to the resulting classification. As a result, we decided that more complex models like CNNs weren't necessary. They're used for more complicated data structures and relationships, which aren't present in the laptop dataset. Our feedforward neural network contains 2 hidden layers; the first with 64 neurons and the second with 32 neurons. Both are followed by ReLU activation functions and dropout layers. Because our project goal was to classify laptop type given its features, the final output layer outputs logits for each of the classes using a fully connected layer. With this model, a validation accuracy of 0.73 was achieved.

To improve the performance of our model we performed hyperparameter tuning on the learning rate, dropout rate, epochs, and batch size. The learning rate controls step size, the dropout rate controls how many neurons are randomly deactivated to reduce overfitting, the number of epochs controls how many passes over the training dataset are performed to stop training once validation accuracy is maximized, and batch size controls how many training examples are processed together in a single forward and backward pass. To select the best hyperparameter values, we used grid search to iterate through all combinations of values and evaluated each one based on its

validation accuracy. This resulted in a learning rate of 0.01, a dropout rate of 0.05, a batch size of 4, and stopping at 60 epochs. We achieved a test accuracy of 0.77.