Predictive Analysis of Computer Hardware Prices through Machine Learning

Linear Regression vs Decision Tree

Andrew, Dimitri, Robby, and Idalia

Our objective for this project is to utilize machine learning to enhance the decision of computer hardware. Specifically, we intend to predict the prices of computer hardware, through a dataset we categorize based on CPU, GPU, RAM, and SSD components. Through the application of linear regression, our objective is to forecast hardware prices, thereby improving the user decision-making. We also included another model which was the Decision Tree model for regression to then compare its results against the results of the Linear regression model to see which would perform the most accurately in computer hardware price prediction.

1. Introduction

When it comes to choosing computer hardware, it is important to consider different factors such as the performance, compatibility, and budget constraints. As the demand for high-performing computing systems continues to rise, our goal is to use machine learning to influence the selection of computer hardware. To enhance the accuracy and efficiency of predicting hardware prices for the benefit of the user. In order to implement machine learning for the performance evaluation, a robust dataset forms the foundation for the analytical framework.

2. Data Set Analysis

Our dataset for our project samples from Dilshaan Sandhu's, Computer Hardware Dataset from Kaggle. Hardware performances will be the features (X) and this will also include the Brand and Components as part of the (X) features, and price which we are trying to predict will be the y, substituting the linear regression equation. Our dataset is organized by different hardware components, such as CPU, GPU, RAM, and SSD. From there we plan to test each component based on different brands. And depending on the components, we will compare each brand with other features. For example, RAM we will consider RAM speed and Memory Size. For GPU we will consider Base Clock, Turbo Clock, and Memory Size. For CPU we will consider Base Clock, Turbo Clock, and Cores. And lastly for the different SSD brands we will consider Memory Size. Our dataset is in the format of a spreadsheet but will be converted into a CSV file. Below is our data visualization:

	Brand	Component	Base Clock(GHz)	Turbo Clock(GHz)	Cores	Memory_Size(GB)	RAM_Speed(MHz)	Price
2	AMD Ryzen 9 7950X	CPU	4.5	5.7	16	0	0	544.02
3	AMD Ryzen 7 7700X	CPU	4.5	5.4	8	0	0	318.44
4	AMD Ryzen 5 7600	CPU	3.8	5.1	6	0	0	216.89
5	Intel Core i9-13900K	CPU	3	5.2	24	0	0	522.99
6	Intel Core i7-13700K	CPU	3.4	5.3	16	0	0	369.99
7	Intel Core i5-13600K	CPU	3.5	5.1	14	0	0	297.99
8	ASUS GeForce RTX 4070 TUF	GPU	1.9	2.5	0	12	0	629.99
9	MSI GeForceRTX 4070ti	GPU	2.3	2.6	0	12	0	705
10	GIGABYTE GeForce RTX 4070ti	GPU	2.3	2.6	0	12	0	719.99
11	Corsair LED 32GB	RAM	0	0	0	32	3200	81.99
12	Corsair Vengeance LED red DDR4-3000 CL15	RAM	0	0	0	16	3000	328.18
13	G.Skill Aegis DDR4 DDR4-3000 8GB	RAM	0	0	0	8	3000	21.99
14	Samsung 980 PRO Series 2000 GB	SSD	0	0	0	2000	0	174.99
15	Samsung 970 Evo Plus 1000 GB	SSD	0	0	0	1000	0	104.95
16	Corsair MP600 Pro LPX 4000 GB	SSD	0	0	0	4000	0	329.99
17	ASUS GeForce RTX 3070 ROG Strix O8G White LHR	GPU	1.7	1.9	0	8	0	779.99
18	Gigabyte GeForce RTX 3050 Gaming OC	GPU	1.5	1.7	0	8	0	295.5
19	Gigabyte GeForce RTX 3070 Ti Master	GPU	1.6	1.9	0	8	0	319.99
20	Corsair Dominator Platinum + AF	RAM	0	0	0	16	4800	79.99

21	Crucial Ballistix Elite series	RAM	0	0	0	4	1600	16.3
22	Corsair LPX 16GB	RAM	0	0	0	16	3000	43.99
23	Corsair MP400 4000 GB	SSD	0	0	0	4000	0	553.45
24	Sabrent Rocket	SSD	0	0	0	512	0	44.99
25	Samsung 970 EVO 500 GB	SSD	0	0	0	500	0	69.9
26	AMD Ryzen 7 5700X	CPU	3.4	4.6	8	0	0	162
27	ASUS GeForce RTX 3080 ROG Strix V2 O10G LHR	GPU	1.4	1.85	0	10	0	499.99
28	Corsair Vengeance Low Profile DDR3-1600 CL9	RAM	0	0	0	16	1600	24.99
29	ADATA XPG SX8200 Series	SSD	0	0	0	240	0	46.9
30	Intel Core i3-12100F	CPU	3.3	4.3	4	0	0	87.04
31	EVGA GeForce RTX 3090 XC3 Gaming	GPU	1.39	1.69	0	24	0	899.99
32	Crucial Ballistix Sport LT white DDR4-2666 CL16	RAM	0	0	0	8	2666	89.96
33	Samsung 860 EVO Series 1000 GB	SSD	0	0	0	1000	0	149.99
34	Intel Core i5-11400	CPU	2.6	4.4	6	0	0	150.95
35	INNO3D GeForce RTX 3060 Ti Twin X2 LHR	GPU	1.41	1.65	0	8	0	269.99
36	G.Skill Ripjaws V DDR4-3200 32GB	RAM	0	0	0	32	3200	69.99
37	Seagate FireCuda 520 2000 GB	SSD	0	0	0	2000	0	187.99
38	Intel Pentium G4400	CPU	3.3	3.3	2	0	0	45.99
39	MSI GeForce RTX 3070 Ti Ventus 3X 8G OC LHR	GPU	1.57	1.8	0	8	0	369.99
40	Corsair Vengeance LPX black DDR4-2666MHz CL16	RAM	0	0	0	8	2666	24.99
41	Corsair Force Series MP500 240GB MLC NVMe	SSD	0	0	0	240	0	54.99

Our dataset includes 10 different brands for each component(CPU, GPU, RAM, SSD). For the total of 40 different items to test. The original data set from Kaggle included way more over 1000 different computer hardware components but because not all were very popular hardware components the everyday consumer would go after on the market, so we simplified our dataset to a sample size of 40 of the popular main brand components that were well known. Since our dataset is designed categorically, we face the problem of our data not being read for the code. Algorithms such as regression work with numerical data. Therefore, we decided to use a One-hot encoder to convert our categorical variables into a numerical representation format.
Here is our line of code using the One-hot Encoder to convert:

```
    one_hot_encoded_data = pd.get_dummies(data, columns = ['Brand',
'Component'])
    print(one_hot_encoded_data)
```

3. Machine Learning Algorithm(s) Description

For our Machine Learning Algorithms, we start off with the approach of linear regression. Linear regression is used to predict the value of a variable based on another, and that is what we are trying to do, predict the prices of each computer hardware based on their features. In our context, the input features represent various specifications of computer hardware components (CPU, GPU, RAM, SSD), while the target variable is the price of the hardware. Before applying linear regression, we preprocess the data to ensure compatibility with the algorithm. This involves converting categorical variables into numerical representations, as linear regression models work with numerical data. We use the One-hot Encoder technique to achieve this conversion. After converting our data for a numerical representation. We then proceed with the data preprocessing. We select relevant features that have potential predictive power in determining hardware prices. These features include specifications such as base clock speed, turbo clock speed, number of cores, memory size, and RAM speed. We create “X” which represents the features or input data. We also have a variable called categorical_features with the list of the columns of one_hot_encoded_data. It selects columns whose names start with either “Brand_” or “Component_”.

```
each_component_features = ['Base Clock(GHz)', 'Turbo Clock(GHz)',
'Cores', 'Memory_Size(GB)', 'RAM_Speed(MHz)']
categorical_features = [
    col for col in one_hot_encoded_data.columns
    if col.startswith('Brand_') or col.startswith('Component_')
]
```

```
X = one_hot_encoded_data[each_component_features +
categorical_features]
X.head()
```

For our “y” it will be the “Price”. We divide the dataset into training and testing sets to evaluate the performance of the model. The training set is used to train the model, while the testing set assesses its predictive accuracy on unseen data. We then want to use sklearn to use the train_test_split.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2)
```

We instantiate a Linear Regression model from the scikit-learn library and fit it to the training data. Subsequently, we evaluate the model's performance using metrics such as Mean Squared Error (MSE). We try to fit the X_train and y_train to our linear regression model. Then, we want to test our training set using the mean_squared_error (MSE). Our result for the Training MSE was 12564.166853173954. We were only able to obtain this value as our training data set result because we had to drop the ‘Brand’ column due to the model was causing us some issues of Overfitting. A lower value of MSE generally indicates a better performance. This shows that it was not performing too well. This is when we then decided to test out the performance of another model. The model we chose to use in comparison with the results of the Linear Regression model was the Decision Tree model. Because our dataset contained categorical features, the Decision Tree model was better suited for this kind of data set. It could handle it a lot better than the Linear Regression model. The process was almost the same as before with data preprocessing. We used the one hot encoder again but this time we did not drop any columns as we did before with the Linear regression model. For the regression task we imported the DescisionTreeRegressor model from the sklearn package. We took our encoded preprocessed data and fitted it to the Decision Tree model.

```
from sklearn.tree import DecisionTreeRegressor
dtr_model = DecisionTreeRegressor()
dtr_model.fit(X_train, y_train)
```

Next, in order to measure how accurate this model can be when it comes to a prediction task we had to calculate the mean squared error. This is the same as before with the Linear regression model when it came to measuring its performance. We calculated the MSE of the training set first to see how well the model can perform on seen data.

```
from sklearn.metrics import mean_squared_error
dtr_train_predict = dtr_model.predict(X_train)
dtr_train_mse = mean_squared_error(y_train, dtr_train_predict)
```

This same process occurred for trying out the test set which is the unseen data to the model.

```
dtr_test_predict = dtr_model.predict(X_test)
dtr_test_mse = mean_squared_error(y_test, dtr_test_predict)
```

As a result, the MSE of the training set passed into the model was 1843.4295736111114 which is a lot better than the training MSE results from the previous linear regression model.

4. Results Analysis

As mentioned before when talking about the accuracy of the Linear Regression model, the case of overfitting was occurring. When we first ran our data through our linear regression model, the training MSE we would get as a result was originally 4.1542598560296313e-26. At first when inspecting this value it actually is telling us that the model is performing exceptionally well. This value shows that the model is very accurate because that MSE is very close to zero and the closer you are to zero, the more accurate the model is. The issue was when we attempted to run the test set data on the same model. Originally, we would get a value above 65,000. This value tells us that the model performed very poorly on our test set data. This issue is what we call the case of overfitting in machine learning when the model overperforms on the training set and underperforms on the test set. In order to tackle this problem, we first decided to scale our data using a MinMaxScaler from sklearn. which scales the features to a specified range (usually between 0 and 1).

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

Our goal for the MinMaxScaler was to see if the model performs well on both the training and testing sets, it indicates that it has achieved a good balance between systematic error and random error, consequently, is capable of making reliable predictions. The results still showed a major difference between the training set and the testing set. This is then when we decided to drop the 'Brand' column as one of our features. After dropping this feature, we got out training and test MSE results to be relatively closer to each other than before.

```
Training mse:  12564.166853173954
```

```
Test mse:  15387.608158948095
```

Now when it comes to the decision tree model, we did not have to make any modifications to our dataset and no data set scaling either. Since the decision tree model handles categorical data a lot better, this is why there was no need for modifications. We just simply ran our training and test sets to the model. These were the results for the training and test MSE's from the decision tree model.

```
Training mse:  1843.4295736111114
```

```
Test mse:  1568.725875
```

5. Discussion and Conclusion

Despite encountering challenges such as overfitting, we were able to show the performances of the Linear Regression model and the Decision Tree model to see which one would be the better model for the prediction of computer hardware prices. As you can see in the results, the decision tree model outperformed the linear regression model by a lot since our dataset was more well suited for the model. You can clearly see a huge difference in MSE's when looking at the Decision Tree model's performance and compare it up against the Linear Regression model's performance. Now, when you carefully look at just the performance of the Decision Tree model, you can see the test MSE is lower than the training MSE. We want this result to occur, to show that the model can perform more accurately on unseen data which is the data that is not used to train the model. Overall, we were able to produce a model performance for both models and we were able to show that we can predict computer hardware prices through machine learning models. In our case, the Decision tree model was the most well suited and most accurate model for our data set of Computer Hardware Components.

6. Statement of individual contribution

For the whole project everyone in the group worked together and contributed a lot. Our group even attended NKU celebration and presented our project together. There was equal contribution, and the project was able to be completed in a timely manner.

Reference

Computer Hardware Dataset –

https://www.kaggle.com/datasets/dilshaansandhu/general-computer-hardware-dataset?select=CPUData.csv