mingjun wei

2202588F

P01

MLDP REPORT

Table of Contents

[Introduction 2](#_Toc142060769)

[About the project 2](#_Toc142060770)

[About the dataset 2](#_Toc142060771)

[1 Data exploration and Pre-processing of data 3](#_Toc142060772)

[1.1 Basic exploration 3](#_Toc142060773)

[1.2 Data cleaning 3](#_Toc142060774)

[1.2.1 Removing unnecessary rows in the data 3](#_Toc142060775)

[1.2.2 Removing unnecessary columns in the data 4](#_Toc142060776)

[1.2.3 Removing Missing data 4](#_Toc142060777)

[1.3 Data engineering 5](#_Toc142060778)

[1.4 Distribution analysis 6](#_Toc142060779)

[1.5 Corelation analysis 7](#_Toc142060780)

[1.6 Feature engineering 7](#_Toc142060781)

[Methods and improvements 10](#_Toc142060782)

[Feature selection 10](#_Toc142060783)

[Mass running models 10](#_Toc142060784)

[Adjusting hyperparameters 12](#_Toc142060785)

[Gradient boosting 12](#_Toc142060786)

[Bagging regressor 14](#_Toc142060787)

[XBGregressor 16](#_Toc142060788)

[Ensemble methods 17](#_Toc142060789)

[Results and analysis 18](#_Toc142060790)

[Conclusion 19](#_Toc142060791)

[References 20](#_Toc142060792)

# Introduction

## About the project

In this project I will be trying to predict the performance of a GPU based on certain specifications of the GPU. The performance will be a set standard and the specifications will be common specifications that all GPUs will contain. This includes things like transistor count, die area, power draw and frequency the GPU runs at.

## About the dataset

The dataset is taken from Kaggle and is titled [CPU and GPU Performances Dataset](https://www.kaggle.com/datasets/michaelbryantds/cpu-and-gpu-product-data). Below is the data dictionary that is important to training the model

|  |  |
| --- | --- |
| Column name | description |
| product | name of the GPU |
| release date | release date of the GPU in year month and date |
| process size [NM] | lithography process of the GPU chip also known as the size of the transistors on the chip we've nanometers as the measurement unit |
| TDP [W] | power draw of the graph is cut with what as the measurement unit |
| die size [mm^2] | area of the size of the chip in millimeter square |
| transistors [million] | count of transistors on the GPU chip in millions |
| frequency [MHz] | frequency of the GPU that it runs at in megahertz |
| foundry. | The foundry where the GPU is made |
| vendor | the company where the GPU is from |
| FP32G flops | this is a performance measurement of the GPU |
| release year | this is the measurement of time between the oldest product release year and the current products release year |
| die density | derived column that is divided from the number of transistors divided by die size to better represent the relationship between these two columns and the target column |
| transistor frequency | derived column from transistor count to the power of 0.5 multiplied by the frequency to better represent the relationship of these two columns and the target column |

# 1 Data exploration and Pre-processing of data

## 1.1 Basic exploration

A screenshot of a computer

Description automatically generatedFirst, I loaded the dataset in and found the different columns and their datatype and the sum of missing values for each of the columns

A screenshot of a computer program

Description automatically generated

The above shows that there are many columns missing for the columns FP16,FP32 and FP64, this is because only rows that have the type value of GPU have values in those columns and since in this project we are focused on predicting the GPU’s FP performance, the rows where type is CPU is not related.

Form this we also can see the numerous number of columns which most of them will not be useful when we are only trying to predict the FP values of the GPUs. There is 4854 rows of data in the dataset and 14 columns which gives it a healthy amount of rows to columns ratio where the chances of models not having enough rows to make out a pattern is minimised.

## 1.2 Data cleaning

### 1.2.1 Removing unnecessary rows in the data

A screen shot of a computer program

Description automatically generatedNext, I proceeded with data cleaning. The first thing that I did was to remove the rows where the column “Type” is CPU since we are not taking CPU into consideration for this project.

To achieve this, I looped through the dataset and found the rows where Type == CPU and dropped them. However, this will cause a mismatch in the index as the index will not automatically be reset to be a continuous ascending value. Thus, I also reset the index before verifying the changes by finding the unique values of the column.

### A black background with white text Description automatically generated1.2.2 Removing unnecessary columns in the data

After that I dropped the columns that I deemed unnecessary to the project. The code used is shown in the left.

A screenshot of a computer

Description automatically generatedIn the dataset there are 3 metrics of measurement of GPU performance. FP32(single precision floating-point format), FP64 (double-precision floating-point format) and FP16 (half-precision floating-point format). These are essentially the measurement of performance just with differing precisions. The most used format is FP32 as it is the most versatile and have a good balance between precision and speed. It provides 23bits of precision and is supported by most modern GPUs while FP16 and FP64 requires a translation layer most of the time thus making it an inaccurate measure of performance. Thus, I chose FP32 as the target column and dropped the unnecessary FP64 and FP16 columns.

[Unnamed: 0] is the default index column in the dataset and is not needed, [Type] only have an unique value of GPU after removing CPU and thus is not necessary any more.

Above shows the data frame after removing the columns.

### 1.2.3 Removing Missing data

A screen shot of a computer program

Description automatically generatedA computer code with colorful text

Description automatically generated with medium confidenceNext, I removed the rows where the column of [FP 32 GFLOPS] is missing as shown on the left. I did this because FP32 is the column that we are trying to predict and if we use any imputation method, we may decrease the accuracy of the model by introducing skewedness into the data. Thus, I just removed those rows where it is empty.

Next, I remove any row with the columns [TDP (W)], [Die Size], [Transistors] or [Process Size] missing as the missing rows often have multiple columns missing and even if the missing columns do not overlap, the maximum percentage the rows make up is 23% which is not a very significant amount to cause any problems for training of the model.

## 1.3 Data engineering

A computer screen shot of a program

Description automatically generated A screenshot of a computer program

Description automatically generated For data engineering, I decided to transform the column of release date into release year and turn it into a number column. I believed that this is important as in the semiconductor space there is Moore’s law which is the observation that the number of transistors in a piece of silicone doubles about every two years. And since the number of transistors greatly impacts the performance, I believed that if there was a column that had a representation of time, it would help the model grasp the pattern better. The reason the time accuracy is decreased into year is because the generation where the GPU is from is better dictated by the year as when manufacturers release the next generation of GPU’s it is always a staggered release with the higher end products like the RTX3090 and RX6900 released first and have around 4-5 months of time in between until the entire generation of product is released.

To achieve this I first removed the NaT(Not a Time) rows in Release date should there be any in the dataset.

Screenshot of the column before changes
column after changes
After doing that I realised that there is 40 rows that had missing time. I then spliced the year from the release date and made it a new column.

However, since year is a very big number as shown on the left it may cause a problem where the machine put too much priority to the later years and cause bias in the prediction and thus it would be better to instead find the difference between the earliest model and the current model’s release date.

I achieved that through the code shown in the bottom and the column after changes is shown on the right with a yellow border.

A black background with white text

Description automatically generated

## 1.4 Distribution analysis

After cleaning the date I plotted histogram of the data columns to check for distribution. Because of the nature of technology products and advancements I assumed that there will be a skewed in distribution of the columns with most of the values being on the lower end.

A group of blue and white graphs

Description automatically generatedBased on this histogram I can say that my assumption was mostly correct with some columns being the exception, thins like frequency of the GPU and the process size. This is because process size is not determined by the GPU but by the manufacturing Foundry of the GPU chip. And since foundries Improves their technology in parallel with the GPU there is not a lot of corelation between that and the other columns.

Overall the dataset is heavily skewed to the right because of the nature of the advancements in technology does not follow the traditional distribution of dataset.

## 1.5 Corelation analysis

A chart of different colors

Description automatically generated with medium confidenceToo better visualise the corelation I used a heat map and the default corelation calculation method as my dataset is made of mostly numeric column that will impact the modelling later on.

For this I expected the top column with the highest corelation to FP32 will be TDP, Transistors and Freq. I was mostly correct with the exception that transistors was the highest followed by die size and then frequency this was outside of my expectation as I thought TDP would have a higher correlation with FP32.

However this still provided many useful insights as it showed me that most of the columns in this headset have a rather strong correlation with FP32 with the lowest correlation amount being 0.51 which is still rather high correlation count

## A chart with red and green dots Description automatically generated1.6 Feature engineering

During this process I also plotted a few of the columns against each other. I plotted process size and transistors in millions against die size and I found that the relationship between the two is not a very linear relationship is more a logarithmic relationship the plot is shown to the left.

This nonlinear correlation would mean that algorithms would have a harder time capturing the relationship between these two and thus have a harder time trying to predict the value of the FP32. Thus I created a new column called dye density and this is derived from the transistor count divided by die size, this column give a better representation and a better linear relationship between this and the target column

A graph of a function

Description automatically generated with medium confidenceThis graph on the left shows the new derived column relationship between transistor density and FP32 performance.

A graph of a transistor

Description automatically generated

A diagram of a frequency

Description automatically generated with medium confidenceNext I plotted frequency against FP32 and transistor count against FP32 (green) using a scatter plot and I found out that the relationship between both frequency and transistor count against FP32 is not a very linear relationship which makes sense because the laws are the Moore’s law exists and is the observation that the transistor count doubles every generation or so and with increased transistor count there will be an increase and in FP32 performance and thus to better relate this to the target column I have deprived a new column called transistor frequency which is trends is the count to the power of 0.5 times frequency off the GPU. The code for this is shown below.

A graph of blue dots

Description automatically generatedThe reason transistor count is to the power of 0.5 is to reduce the derived column dependency on the transistor count because the transistor count is measured in millions and transistor count on a chip can in the billions so the number for the column of transistors would be a lot higher than the column for transits from frequency in MHz and thus by doing so I can reduce the scale of the column meanwhile ensuring that both transistor in millions and frequency have about the same effect on the column without one overpowering the other. The scatter plot for the new derived column is shown on the left in blue.

A screenshot of a chart

Description automatically generatedCurrently the correlation matrix looks like this shown on the right with a green border and if dictate dictionary is in a table below with all of the columns listed.

|  |  |
| --- | --- |
| Column name | description |
| product | name of the GPU |
| release date | release date of the GPU in year month and date |
| process size [NM] | lithography process of the GPU chip also known as the size of the transistors on the chip we've nanometers as the measurement unit |
| TDP [W] | power draw of the graph is cut with what as the measurement unit |
| die size [mm^2] | area of the size of the chip in millimeter square |
| transistors [million] | count of transistors on the GPU chip in millions |
| frequency [MHz] | frequency of the GPU that it runs at in megahertz |
| foundry. | The foundry where the GPU is made |
| vendor | the company where the GPU is from |
| FP32G flops | this is a performance measurement of the GPU |
| release year | this is the measurement of time between the oldest product release year and the current products release year |
| die density | derived column that is divided from the number of transistors divided by die size to better represent the relationship between these two columns and the target column |
| transistor frequency | derived column from transistor count to the power of 0.5 multiplied by the frequency to better represent the relationship of these two columns and the target column |

# Methods and improvements

## Feature selection

In order to have a better prediction I choose columns TDP, Die Size, Transistors, Frequency, Release year., Die Density and Frequency Count. This is selected based on the correlation score. That is shown above. Before this section.

I then split the data set into X&Y. With the code showed below.

A computer screen shot of a black screen

Description automatically generated

## Mass running models

I ran an array of Very diverse models in order to have a better. Gauge on which model is better. I use a total of 22 models and from these 22 models I'll be selecting the models based on their mean absolute error for testing and training to check for any overfitting or underfitting of data. Overall, I will be picking the one with the lowest Mean absolute error and the ones with the least variance between testing and training Mean absolute error data.

To prevent overfitting as much as possible I used Kfolds with n\_splits = 10 which means that the original training dataset will be further split into 9 training and one testing data, this prevents the model from overfitting by having 10 different runs to pick out the best. I set the random state to my admin number so it can be replicated. The code is shown below.

A screen shot of a computer program

Description automatically generated

The result of this is shown in a table in the Next page.

A screenshot of a black table

Description automatically generated

Based on this, the best performing model based on validation mean absolute error is XGB regressor and begging regressor, however, XGB regressor have a very big difference between the training mean absolute error and the validation mean absolute error which suggest heavy overfitting of the data. Thus, when adjusting paramitas I will be mainly trying to decrease the difference between training MAE and validation MAE, I will also be using the gradient boosting regressor as it has one of the best balances between the difference in training me and validation MAE while keeping a Relatively low making it one of the most accurate models that most likely does not have overfitting.

## Adjusting hyperparameters

### Gradient boosting

For gradient boosting I choose five para metres and these five parameters are shown in the table below, the param grid and the result of the param grid is also shown below.

| **Parameter** | **Description** |
| --- | --- |
| N estimators | Specifies the number of decision trees that are used in the gradient boosting. |
| Learning rate | Controls the contribution of each tree in the final prediction. |
| Max depth | Controls the maximum depth each decision tree in the model can go. The deeper the tree, the more complex the relationships and can lead to overfitting. |
| Min sample leaf | Specifies the minimum number of sample trees required to be added to a leaf node of the decision tree. The larger the number, the more unlikely that gradient booster will be overfitted. |
| Max features | Specifies the maximum number of features when splitting a node. A smaller value can reduce variance and prevent overfitting. |

A screenshot of a computer program

Description automatically generated

A screenshot of a computer screen

Description automatically generated

The table above is the result from the four runs of grid search that I did the difference is calculated from the testing MAE minus sync of the training MAE the higher the difference the more likely that the model has been over fitted. Because I am trying to reduce overfitting and increase accuracy the best-case scenario is the difference between MAE for testing and training MAE is as low as possible while still maintaining testing MAE relatively low value.

I started with a very broad range of parameters and narrowed down based on the trends that I see one of the trends that I saw that was when I reduced the Max depth, the difference between testing MAE and training MAE decreased. When I increased the N Estimators, the testing MAE decreased but when it went over 430, the testing MAE increased again and enlarged the difference between the two MAE.

In the end I ended up with the parameters from the final run as I believed that the MAE for testing is still within an acceptable range while having a small difference between the testing and training MAE. I validated this again with the graph shown below.

A graph of blue and red lines

Description automatically generatedThis is a line plot of the predicted values and the ground truth stacked on each other with the ground truth being blue. This graph shows that the model is able to predict the values rather well and being mostly accurate as they overlap each other quite well.

Bagging regressor  
For gradient boosting I choose five para metres and these five parameters are shown in the table below, the param grid and the result of the param grid is also shown below.

| **Parameter** | **Description** |
| --- | --- |
| N estimators | Specifies the number of decision trees that are used in the gradient boosting. |
| min\_samples\_split | specifies the minimum number of samples required to split an internal node. A larger value can prevent overfitting |
| Max features | Specifies the maximum number of features when splitting a node. A smaller value can reduce variance and prevent overfitting. |
| bootstrap | Bootstrap samples are random samples with replacement from the original dataset, and can help reduce the variance of the model. |

A screen shot of a computer program

Description automatically generated

Above is the table for the 3 runs of grid searchA close-up of a pink background

Description automatically generated I also started with a rather large range of values for each of the para metres and then I slowly narrow them down the goal is still the same as the previous model with minimum difference between the train ma E and the test ma E and as much as possible have the lowest MAE for the models.

Having a low difference in the MPE ensures that there is no overfitting of the model and having a low testing me and trusted a model does well with unseen data sets and having both means that you will perform well with unseen data sets and at the same time there will not be overfitting of the data.

I have increased the Max samples to a rather large value of 0.8 as vendor Max sample value is less than one each of the best estimator will be trained on a random subset of the training set this helps reduce overfitting and improve generalisation performance of the model.

### XBGregressor

| **Parameter** | **Description** |
| --- | --- |
| N estimators | Specifies the number of decision trees that are used in the gradient boosting. |
| Learning Rate | controls the contribution of each tree in the final prediction |
| Max Depth | controls the maximum depth of each decision tree in the model, the deeper the model the better it captures relationship of the data but the more it will tend to overfit |
| Subsample | specifies the fraction of samples to be used for each tree |
| Colsample bytree | specifies the fraction of features to be used for each tree |
| gamma | specifies the minimum loss reduction required to make a further partition on a leaf node of the tree, which means that if there is no significant improvement of the data the gamma threshold will stop the branch early |
| Reg alpha | parameter specifies the L1 regularization term on the weights, this encourages the weight to be more spread out |
| Reg lambda | parameter specifies the L2 regularization term on the weights, encourages the weights to be smaller |

A computer screen shot of a program code

Description automatically generated

in this model the first run of the model gave me a value that I was satisfied with the MAE of this model was 348.5634 and the training MAE on this model was 160.9998 this means that although it is not very accurate on unseen datasets compared to when it started, it greatly reduced the overfitting amount as the difference between the two MAE decreased by over 80. The best para metres of this were 150 estimators and with a L2 regularisation of 0.5 this encourages the model to spread out the weights further by encouraging the weight to be smaller and having less estimators means that there are less trees so the model will have lesser chance to overfit. Plus, I am rather satisfied with this single run of grid search.

## Ensemble methods

after adjusting the hyperparameters the begging regressors still have the tendency to overfit while being more accurate while the XGB regressor is less accurate but have a lesser tendency to overfit thus I decided to use a voting regressor with the optimum hyperparameters found above as the voting regressor would combine both of these regressors together and come up with a new model that would hopefully combine the two regressors advantages.

A screen shot of a computer program

Description automatically generated

Above is the code snippet I used for the voting regressor I have used the optimum hyper parameters that I have achieved throughout the great search done above before this.

A black screen with white text

Description automatically generated

However, the result was not what I expected I had expected the and the E score to be lower and the training MBE score to also be higher, as I had thought that combining both of these regressors together will give me a more accurate model while still maintaining the tendency to not overfit.

# Results and analysis

For the final comparison and analysis I have decided to use gradient boosting regressive the voting regressor An A plane linear regressor I have decided to use these three because the first two were found after comparison with numerous models as they were what I thought to be the best balance of both having good accuracy on unseen data sets while not overfitting to the scene datasets I also wanted to compare this to linear aggressors.

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Test MAE | Train MAE | Difference |
| Gradient Boosting | 348.704 | 158.997 | 189.708 |
| Voting regressor | 297.828 | 82.405 | 215.422 |
| Linear Regression | 734.278 | 739.672 | -5.394 |
|  |  |  |  |

A graph of a voting graph

Description automatically generatedA graph with red and blue lines

Description automatically generated

A graph of blue and red lines

Description automatically generated

The plot above shows the predicted value plotted over the actual value or the ground truth the ground truth is in blue and the predictor value is in red for linear regression you can tell that the rate stands out a lot and you can tell that the rib often overshoots or is below the ground truth this shows that the linear regression is not very accurate because it is often inaccurate a more accurate model would look like in the voting regressor where there are minimum red that is seeping through the blue however this doesn't tell the whole story because if the blue and the red matches exactly or very closely it is a sign of overfitting which means that the model will not perform very well on unseen and datasets that is not trained on both of these can be confirmed by the testing mean absolute error score and training mean absolute error score.

The linear regression have the highest mean absolute error for both training and testing at 734 and 739 this means that the linear regression is not very accurate and doesn't capture the relationship of the data set very well

the voting regressor have the lowest testing mean absolute error this means that the voting regressor is very accurate at predicting however the voting regressor have the largest difference between testing mean absolute error and training mean absolute error. This suggests that the voting regressor would not perform well on values that it has not been trained on or it has not seen.

The gradient boosting regressor performs the best out of all of them by elimination since the gradient boosting regressor has I rather accurate testing mean absolute error score of 348 but it also has one of the lowest differences between mean absolute error for testing and mean absolute error for training at 189. Well, the gradient boosting regressor might not be the most accurate regressor it performs rather well on unseen data sets which is important because we're trying to predict new GPU's that the data set does not include and therefore the model have not seen thus, it is very important that the model that I choose does not overfeed on the training data and I believe that the gradient boosting regressor does that the best

# Conclusion

To conclude, after using a variety of models and training parameters I found out that gradient boosting regressive was the best choice for me because on one hand I wanted to ensure that the accuracy of the model on unseen data set and on the other hand this means that I will have to minimise overfitting and in order to minimise overfitting there should be as less difference between the testing mean absolute error and the training mean absolute error as possible. After looking at the table above that compares the testing MAE training MAE and a difference between the MAEs.

In order to ensure fair testing and comparison whenever there was randomness involved like cross validation I would use a set random seed which is my student admin number to ensure that when this is reproduced at the very least the mean absolute error would be around 5% of what I get this also ensures fairness because every single model is tested under the same conditions with minimal randomness involved. I tested a variety of models some models they are better at Polygon new equations like lesso add some models that combine each other like the voting regressive and the gradient boosting regressor. At the end of the day the gradient boosting regressive stood out

is still up because of it relatively low casting mean absolute error and the lowest difference between the testing and mean absolute error and the training mean absolute error this ensures that the prediction is as suited as it can be for unseen data sets.

Task by using this model in my application it allows users to have a more accurate and reliable prediction every single time they run no matter the values used.

# References

https://docs.google.com/spreadsheets/d/1TBLmA4XnHI9uAti5CoHNRrK1hrM-ZNXjnH0bAAlOJUU/edit?usp=sharing