COMP4702 - FINAL EXAM

Aaron Lim (46420763)

Introduction

In this report, I aim to train the three different supervised machine learning classifiers, K-Nearest Neighbours, Gaussian Naive Bayes and Decision Trees on the given dataset to be able to predict if a cpu is one of the three categories: desktop, server, mobile/embedded and laptop given the values of its features. After all models are trained, I wish to compare these classifiers and determine which one is the best for the given dataset.

Cleaning the Data

The data isn't perfect and there are some instances of incomplete cells and duplicate rows and in order to get more accurate results, we must clean the data.
First I noticed that the column 'cpuName' contains the names of the cpus and wont do much for the classifiers that will be implemented, and will be dropped as well as 'socket' and 'testDate'. There were also rows that contained null values and were dropped as well.
Since we are only interested in classifying Desktop, server, laptop mobile embedded cpus, all other classes were dropped. These included unkown, (Desktop, Server), (Desktop, Mobile/Embedded), (Server, Mobile/Embedded) etc. We get the following populations of labels:

```
            Labels  Frequency
0           Server        634
1          Desktop        820
2           Laptop        401
3  Mobile/Embedded         15
```

After looking at the populations, Mobile/Embedded seemed to have a lot less observations that the other classes and was dropped as training to classify Mobile/Embedded won't give accurate results since the training set will be too small. And we have 8 features:

```
'price', 'cpuMark', 'cpuValue', 'threadMark', 'threadValue', 'TDP', 'powerPerf', 'cores'
```

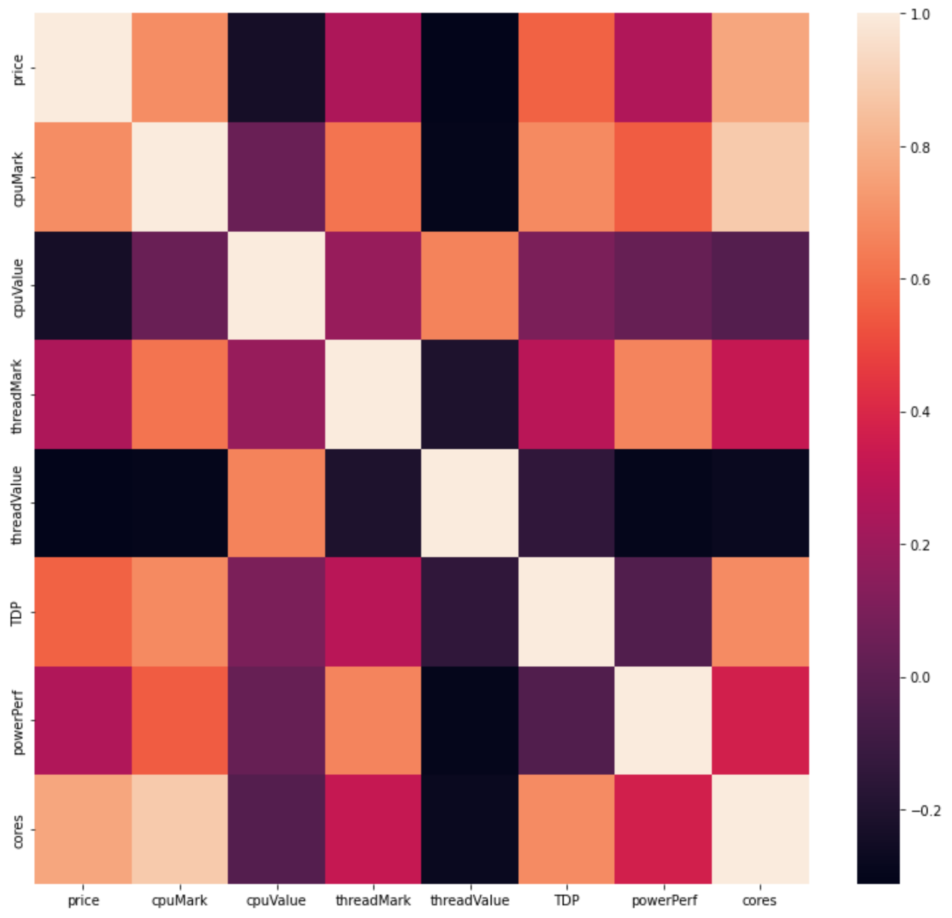
Identifying Classes and Features

The data was split into two datasets, X and y with X being the values of features and y being the classes

Correlation Heatmap

We can first get an idea of how the features of the data interact by looking at the sample correlation heatmap. In the graph, the amount of correlation between features is indicated by the legend on the right.

Correlation Heatmap Between Features



By looking at the heatmap we can see a few correlations between features such as the number of cores correlates with cpuMark, cores with price, cores with tdp.

Principal Component Analysis (PCA)

Objective

In order to apply ML models on the dataset, I will use principal component analysis in to reduce the dimensions (features) that the given data has while still retaining most of the information. This will help with the visualization of the data by allowing us to plot in 2D and to improve running times on machine learning models, especially when the dataset has many features. In order to implement PCA, I used the python library sklearn.

Data Preprocessing

Since the principal components rely on the variance of features, it is important that we first standardize the data. This is because features have different metrics and different scales that describe the data point and thus the variance will be large just due to the nature of the feature.
For example, the variance for powerPerf of a cpu is much larger than the variance of the threadValue. These two features are just at a different scale and would not be fair to compare these variance as it just doesn't make sense. Normalizing the data would give all features a level playing field and the model will not favour a particular feature just due to its scale. I standardized the data using sklearn.

Finding the Best Projections

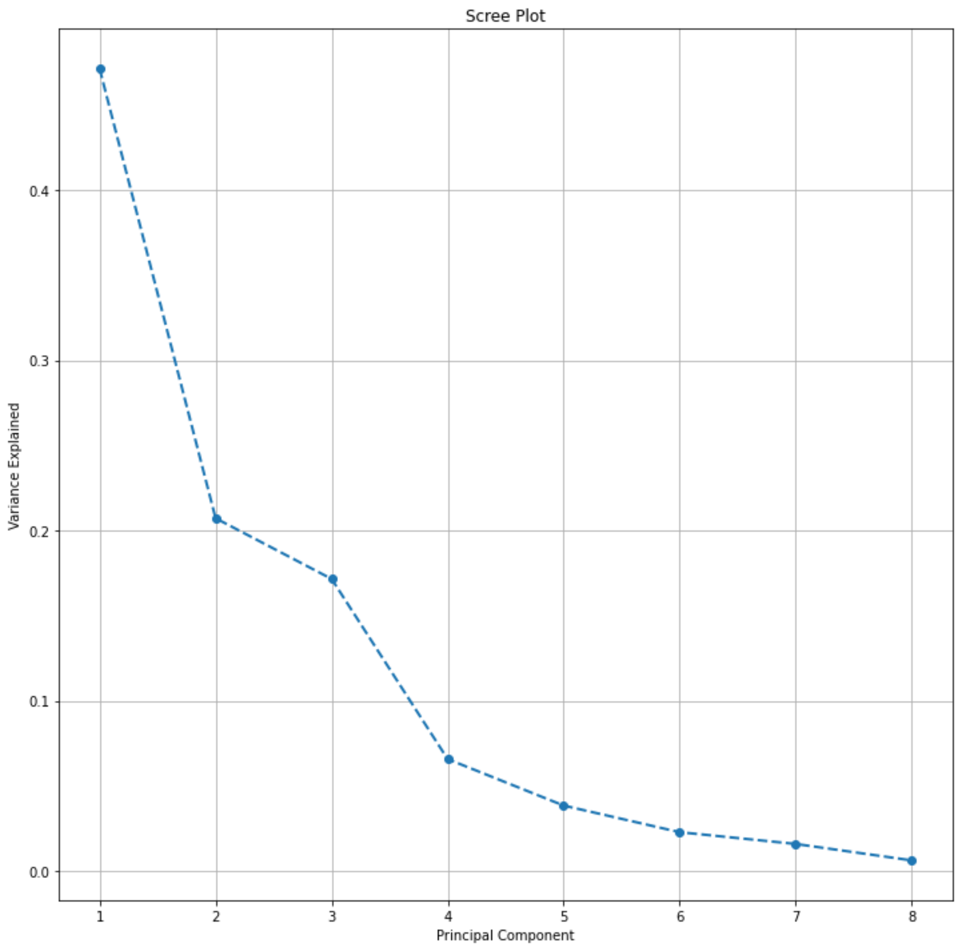
The best projections were found by finding the eigenvectors and values of the covariance matrix which gives us directions that maximize the variance of the data. Projecting our data onto these eigenvectors gives us the principal components. The order of the principal components was done by sorting the eigenvectors by their eigenvalues, going from largest to smallest.

Choosing Number of Components

The number of components is a hyper-parameter for the PCA model where we choose the n-best components based on their total 'explained variance ratio'. Explained variance ratio is a number given to a principal component which is the percentage of variance of the dataset that the component contributes to. Ideally we want less components than the original dataset whilst still keeping most of the variance in order to retain most of the information.

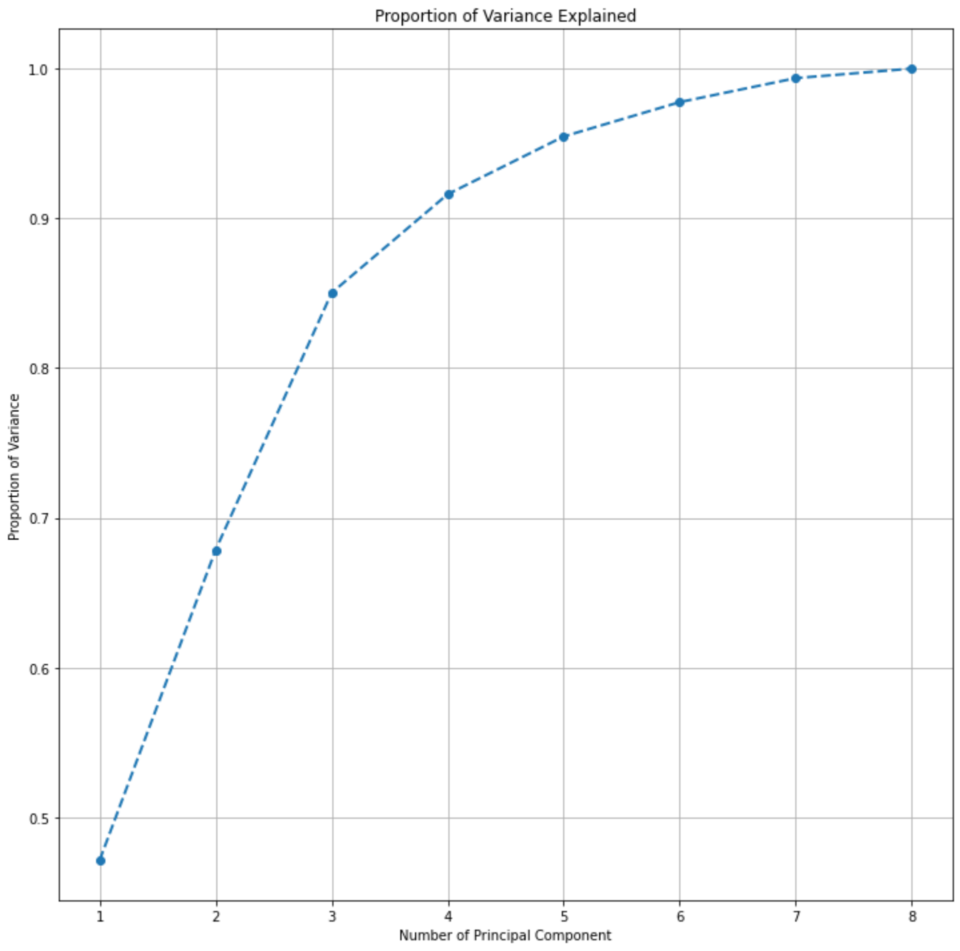
One way to determine how many components to choose is by using a scree graph which is the graph of eigenvalues against their corresponding component.

PCA was run with n = 8, the number of features in X, and all principal components were graphed against their explained variance ratio which gave us this scree graph of our PCA reduced dataset:



We want to choose the 'elbow' of the graph, the point where the addition of more components, does not change the overall explained variance by a significant amount. It's hard to tell where the 'elbow' is but it seems to be around n = 4.
Using 4 components accounts for 91.57% of the variation in the dataset with only 4 components less (50% of the data reduced). Can we do better?

If we plot the total proportion of explained variance for each number of components we can choose, we get this graph:



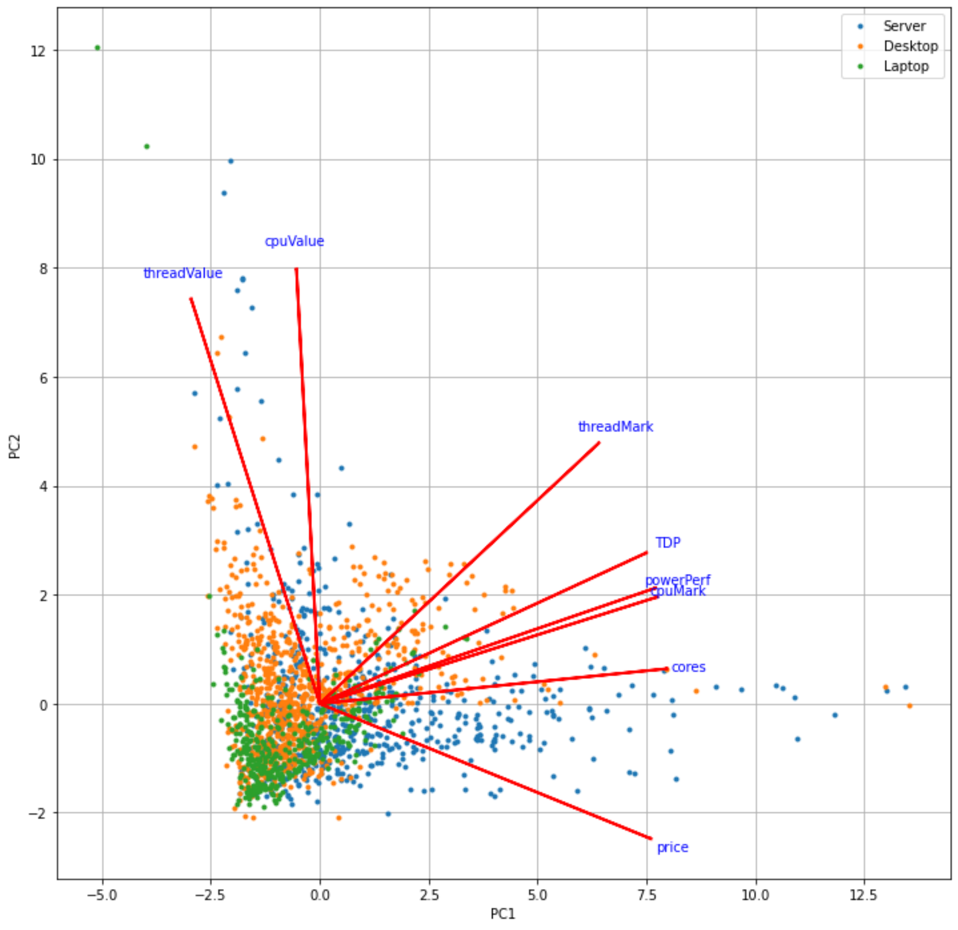
And you can see on the graph that using 4 components gives us 91.57% of the explained variance of the dataset. Which is decent but not the best. So 'eyeballing' the number of components from the scree graph is not the most accurate way of finding the best number. To get a better proportion, we can fix it to an arbitrary number. Here I fixed it to 95% which the corresponding value of n can be found on the graph To obtain a 95% proportion of explain variance, we require to use 5 components which is 3 less. We can now feed this dataset into our machine learning models.

However, if we want to visualise our data, we choose 2 components which allows us to graph on a 2D plot.

Interpreting Graph

Graphing our best component against our second best component we get:

PC1 vs PC2



Using 2 components only accounts for only 67.6% of the variation in the dataset which is fairly informative but definitely not the whole picture. The PC eigenvectors contain values that correspond with how important a feature is in describing that PC with the farther away the value from 0 is, the better the feature is explaining the PC. I plotted all features as vectors to give us an idea of what each PC represents. Vectors that point in the same direction are highly correlated and the closer a vector is to an axis (A principal component) the better it is in representing that PC.

By looking at the graph we can see that the features cpuMark, cores, price, TDP and powerPerf are highly correlated and are the most important in describing principal component 1. Likewise, we can see the features threadValue, cpuValue are highly correlated and are the best in describing PC2.
We can also see that servers tend to have more cores, cpuMark and price than the other cpu types and datapoints with large values in all features tend to be server cpus.
Desktop and Laptop cpus seem have a tight spread and are mostly in the 3rd quadrant suggesting these classes tend to have low value features, with desktops having a bit more spread.
We can see that cpus follow the intuition that servers generally cost more, followed by desktops and then laptops and that servers have more cpuValue/threadValue followed by desktops and then laptops.
We can also see threadMark about a equal distance from both PCs which suggests that it is at equal importance in describing these PCs.

K-Nearest Neighbours (K-NN) Classification

Objective

K-Nearest Neighbours (K-NN) is a supervised machine learning model that aims to classify a given data point based on the classes of the K 'nearby' data points. The objective is to find the K that predicts the cpu type of a datapoint given its features with the most accuracy.

Data Preprocessing

Since K-NN relies heavily on the distances between points, its important that we first standardize the data since the features in the data-set have different scales and features with larger scales greatly effect the shortest distances between points, which standardizing fixes.

Choosing the best K

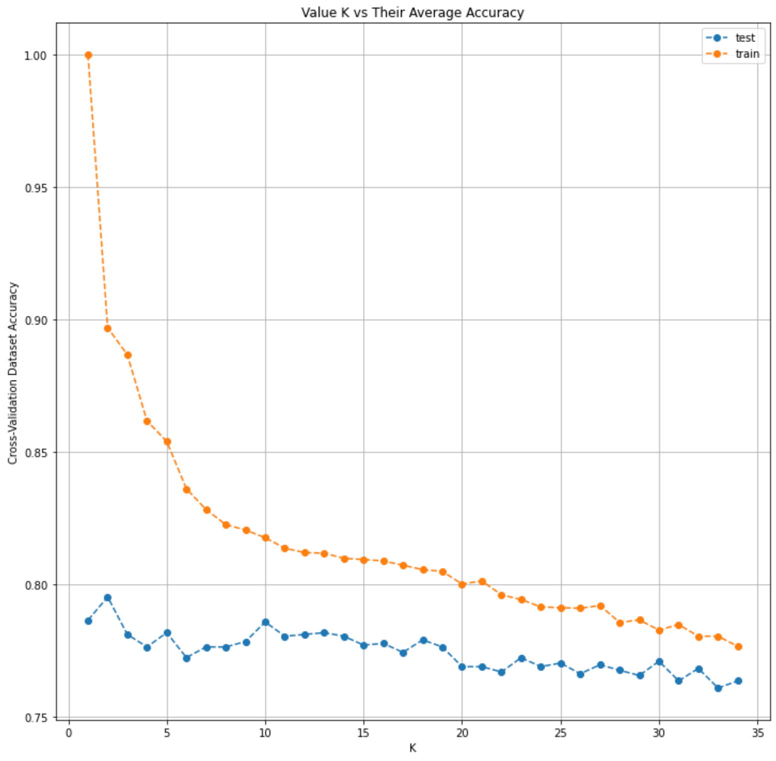
For the best result, we wish to find a K that isn't too low that it could be affected by outliers in the data or too large that classes with not a lot of training samples will always out voted by bigger classes. In general we want a K that works well when we feed the model an unseen data point.

In order to find a K that generalizes the dataset so that it works well for unseen data, I used 10-fold cross-validation which reduces the influence that the chosen training set has on our chosen value of K.

I split the data into Training and Test datasets in a 80-20 split where the training dataset is further split into 10 equal partitions which we will use for our 10-fold cross-validation. Before the 80-20 split, the ordering of the dataset was shuffled so that the values in the training and testing datasets aren't influenced by the initial ordering and that the datasets have a diverse set of classes to train and test on.

The general idea is we choose a partition to be our cross-validation partition and we train our model on the other partitions and find the accuracy of predicting the cross-validation partition. This is repeated 9 more times choosing a different partition to be our cross-validation and we do this for a range of values of K. We take the average cross-validation accuracy for each value of K and choose the highest value to be our K. and then evaluate on the test dataset. This process was implemented using sklearn's KFold function.

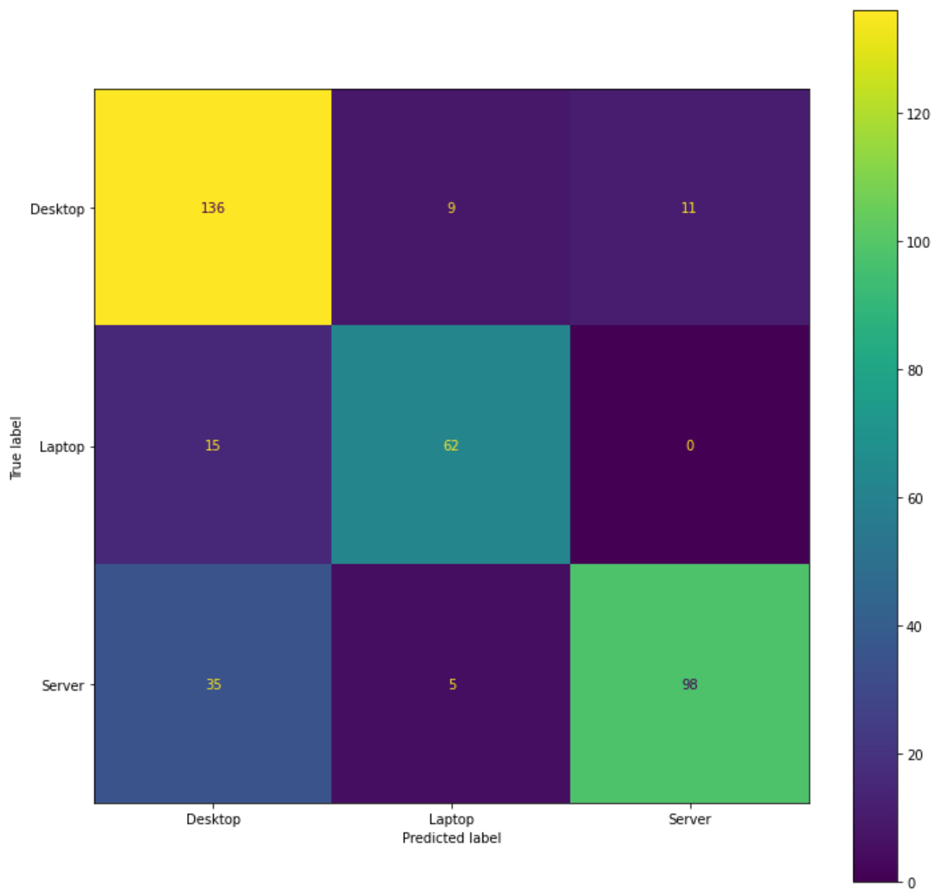
I arbitrarily considered the range of Ks to be from 1 to 35 and ran 10-fold cross validation wtih K-NN being implemented using the sklearn library with uniform weights. The following graph depicts the average cross-validation test accuracy score for each value K.



We can see that accuracy peaks for K = 2, and for larger Ks, accuracy decreases. I took the value of K that gave the highest average test accuracy which is K = 2 which had 79.52% average accuracy.

Fitting the Model and Evaluation

We then train the K-NN model with K = 2 on the whole training dataset and get a final evaluation on the accuracy of our model by checking the prediction accuracy on the test data. The accuracy on the test dataset was 79.78% and 89.89% on the training set. So now we have a classifier for the dataset. The overall performance can be seen on the following confusion matrix:



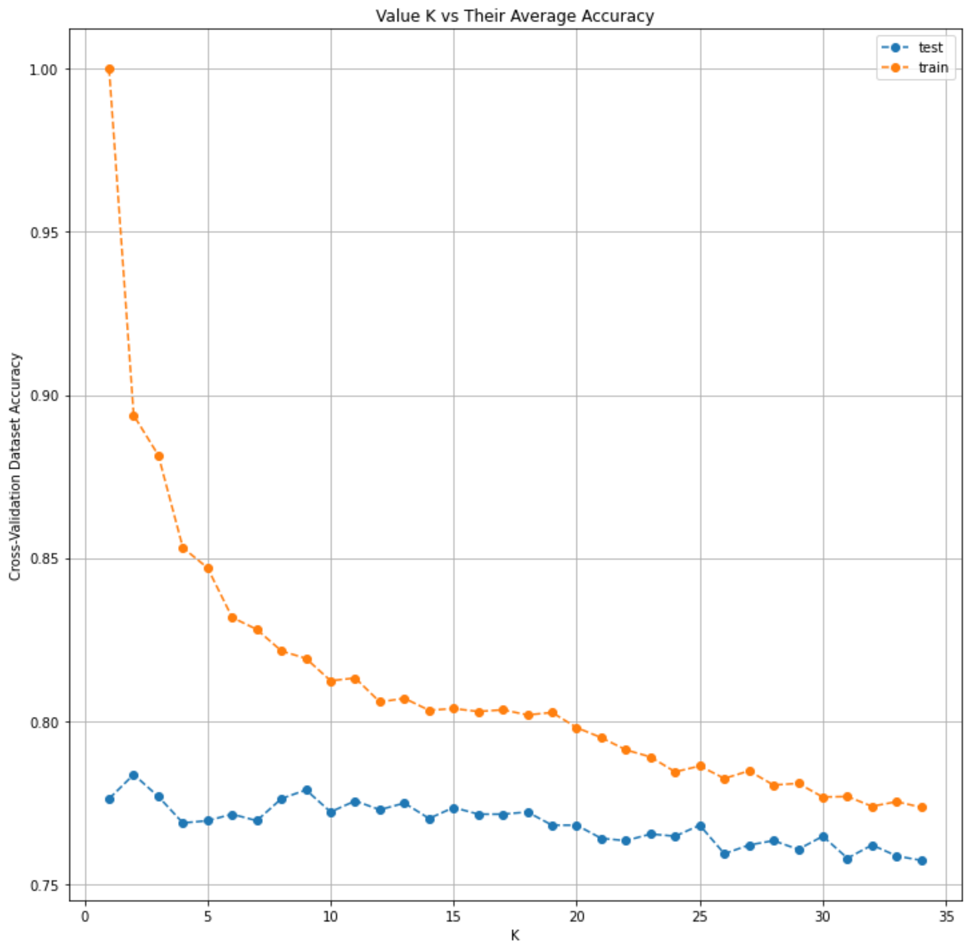
And its accuracy for each class can be seen here:

```
            Labels  Test Accuracy
0           Desktop    0.87179487
1           Laptop     0.80519481
2           Server     0.71014493
```

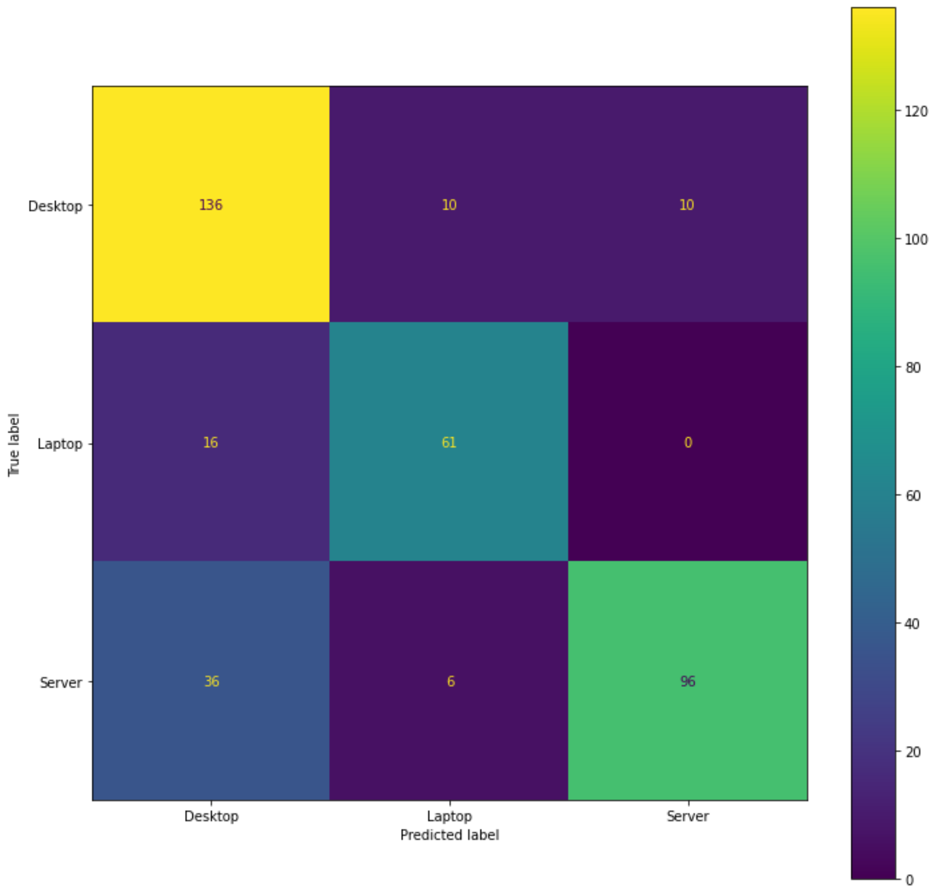
We can see that this classifier works pretty well for classifying desktop cpus, okay for laptops but quite poorly for servers. We can see in the confusion matrix that this K-NN model that it predicted a server as a desktop 35/138 of the time.

Running With PCA

I wanted to see the effect PCA has on K-NNs performance to see if would run any better as it gives features more variability. I run PCA for the data set with a fixed proportion of variance of 0.95. The following is the results from the 10-fold cross validation.



We can see this runs pretty similarly to K-NN. The K chosen was K = 2, and was trained on the whole training set and had 78.36% average accuracy, 78.97% test accuracy and 89.62% training accuracy which is a little bit less than without PCA.
We can also evaluate its performance using the confusion matrix:



And its accuracy for each class can be seen here:

```
          Labels  Test Accuracy
0         Desktop    0.87179487
1         Laptop     0.79220779
2         Server     0.69565217
```

The main difference is in the accuracy in laptop and server which has decreased. This may have been due to a small loss in information when removing 3 PCs.

Gaussian Naive Bayes (GNB)

Objective

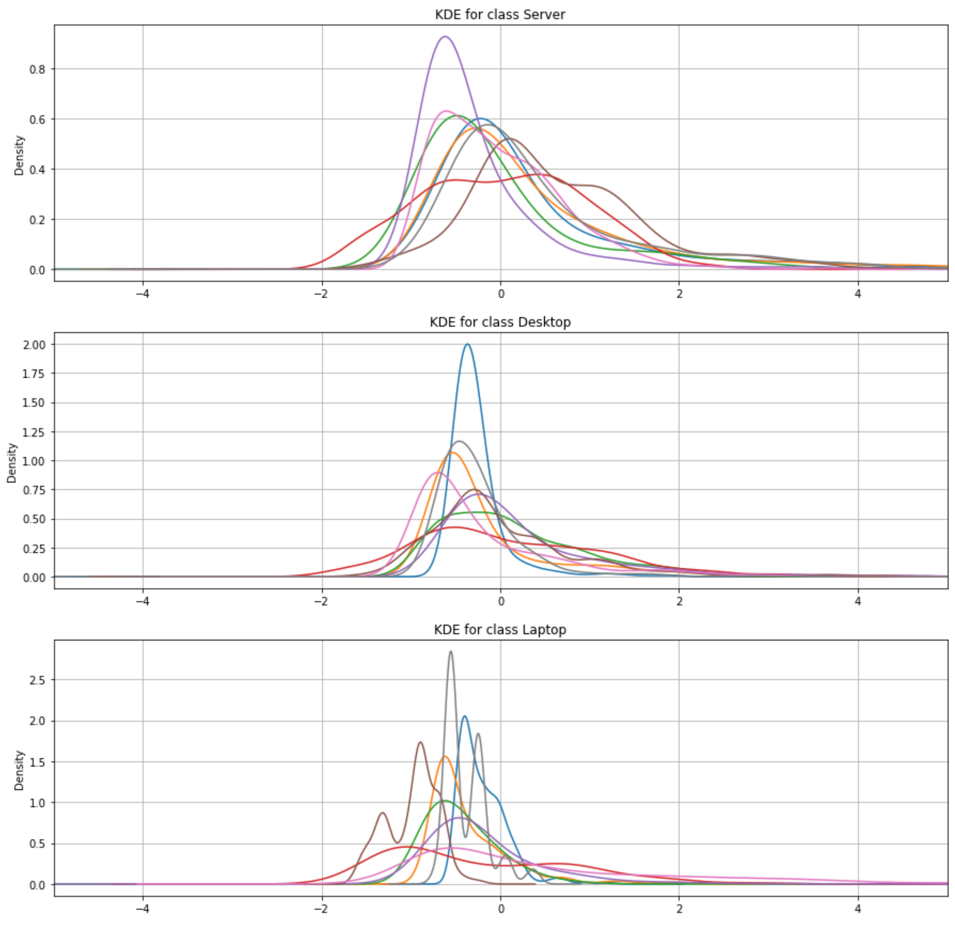
Gaussian Naive Bayes is a machine learning classifier that works under the assumption that the classes of the dataset follow a Gaussian distribution and classifies a given datapoint to the class that gives the highest probability. The objective is to use GNB to predict the cpu type of a datapoint given its features. GNB was implemented using sklearn.

Checking Assumptions

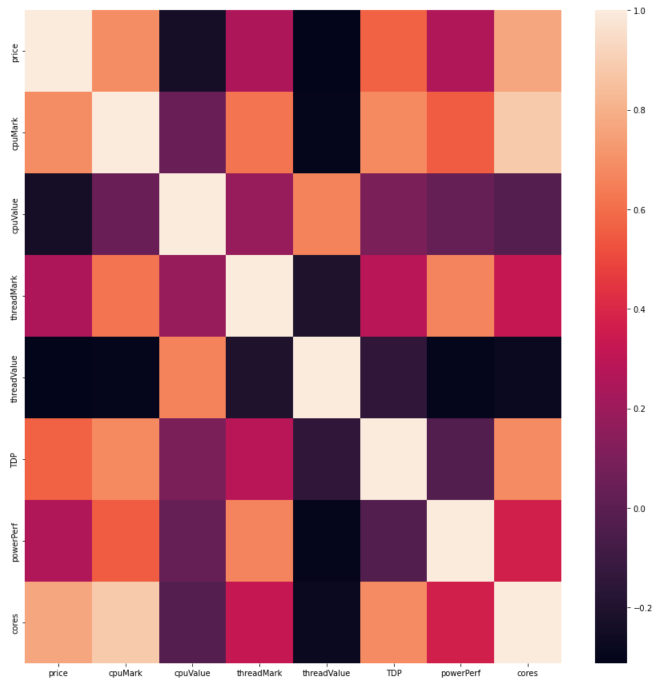
Before applying Gaussian Naive Bayes, we should check our assumptions such as:

- Each class follows a Gaussian distrubtion
- Features are independent
- Classes are dependent on features

In order to check each class's likelihood distribution, we can use apply kernel density estimation for all features for each class and graph the results. The KDE used here uses the Gaussian distribution as kernels. The bandwidth length is automatically chosen by the scikit KDE model which uses "scott's rule".
The data is standardized before plotting so that the distributions center will be at 0 so we can see clearly if a distribution looks weird (not centred at 0). Here are the graphs of the kdes of all features for each class.



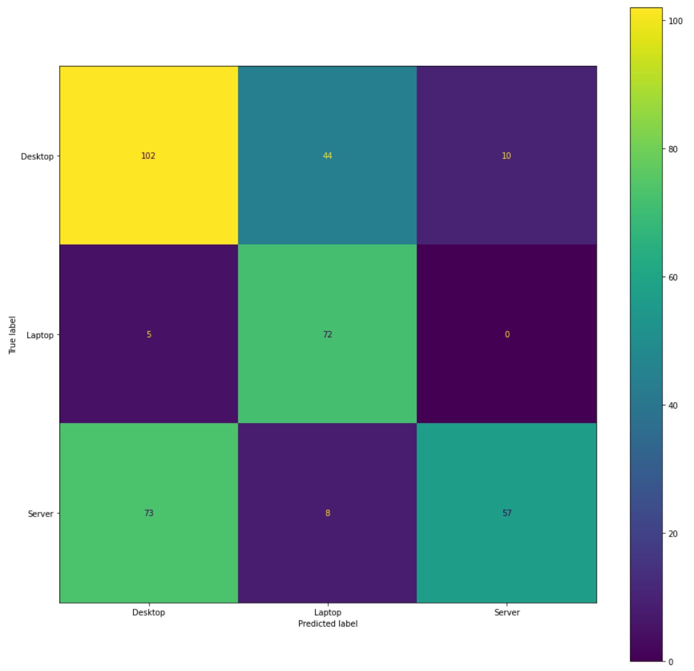
And by inspection, we can see that the features look approximately normal distributed. We can also check the independence of features by looking at the sample correlation heatmap. The differences in distribution between classes also indicate that classes are dependant



As you can see, there is quite a lot of correlation between features which breaks one of our assumptions, but we will continue to use this method and see why this assumption is important.

Fitting The Model and Evaluation

The data was shuffled and then split into Training and Test datasets in a 80-20 split and then trained on the Gaussian naive Bayes model implemented by sklearn.
The model was then tested on the test dataset and gave a 62.26% test accuracy and a 62.12% training accuracy. The confusion matrix is given by:



We can also see the test accuracy for each class:

```
            Labels  Test Accuracy
0          Desktop     0.65384615
1           Laptop     0.93506494
2           Server     0.41304348
```

You can see that the performance for classifying laptops is much better than classifying the others and overall when compared to K-NN's performance. However the test accuracy for the other classes are very low. By looking at the confusion matrix we can see that the model is having a hard time classifying desktops especially between laptops. Classifying severs has a similar problem and is having a really hard time differentiating between servers and desktops, infact it is classifying servers as desktops most of the time. So this model has great laptop classifying accuracy, but really bad performance on the other classes. This test accuracy may be due to the violation of the assumptions of feature independence. Specifically, the high correlation in features.

Decision Tree Classifier

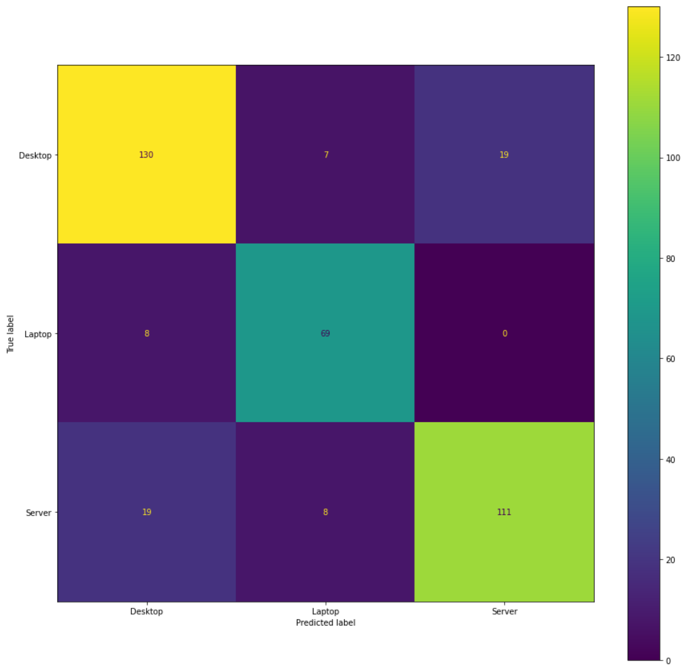
Objective

Decision Trees is a classifier that aims to finds decision boundaries of a datasets features that create decision regions in which we can classify datapoints that lie in these regions. The objective is to find the best Decision Tree that classifies the cpu type given the features of a datapoint. Trees were found using DecisionTreeClassifier implemented with sklearn with uses the gini index as a measure of impurity.

Finding a Preliminary Decision Tree

To find a good decision tree, we first train a decision tree that would use to be our basis so that we can prune this tree to get better test accuracy. Such tree would be the decision tree of our dataset that has full depth meaning that all leaf nodes are pure (perfectly classify training data). The data was shuffled and a 80/20 training/testing split was made on the data and then used to train our decision tree classifier, using gini index as a measure of impurity.

This baseline decision tree has a test accuracy of 84.36%, a training accuracy of 100%, 240 leaves and 16 depth. Its performance can be summarized with the following confusion matrix.

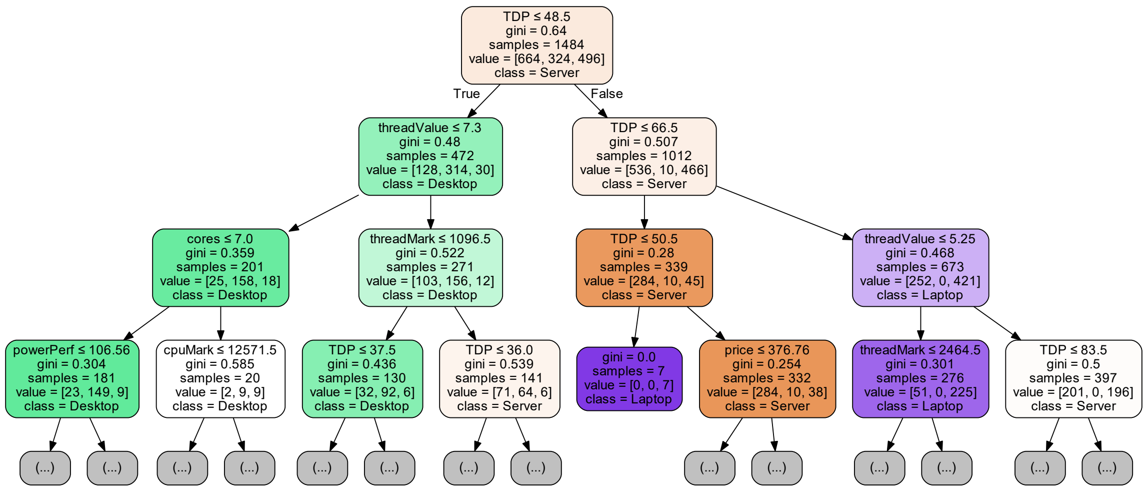


We can also see the test accuracy for each class:

```
           Labels  Test Accuracy
0         Desktop     0.83974359
1          Laptop     0.88311688
2          Server     0.82608696
```

We can see that this classifier runs much better across all classifiers. It still runs into the problem on classifying servers as desktops but this problem is reduced.

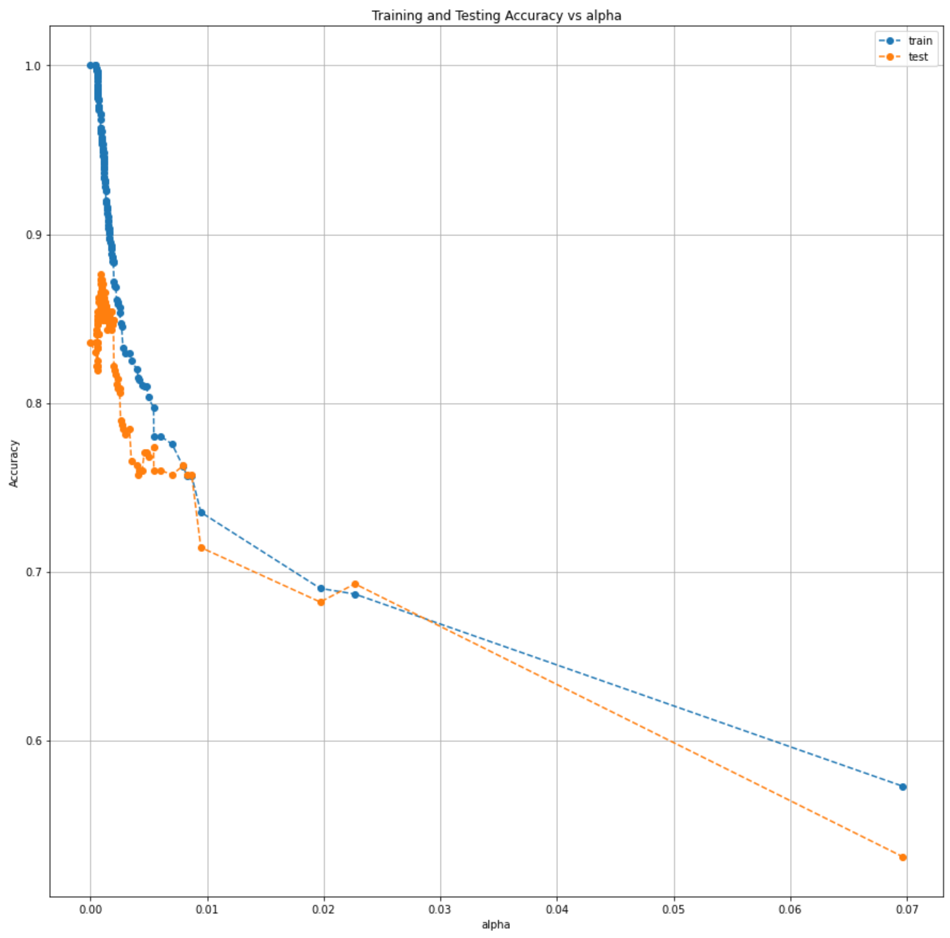
We can get an idea of what this tree looks like by graphing the tree up to a depth of 3:



The majority class is specified by 'class' and also by the color of the node.
Since this preliminary decision tree runs until all leaf nodes are pure, this leads to 100% accuracy on training set but would have dubious results on the test set as this might have over-fitted the training data. We wish to find now if we can do better by pruning, that is, can we remove nodes to improve test accuracy.

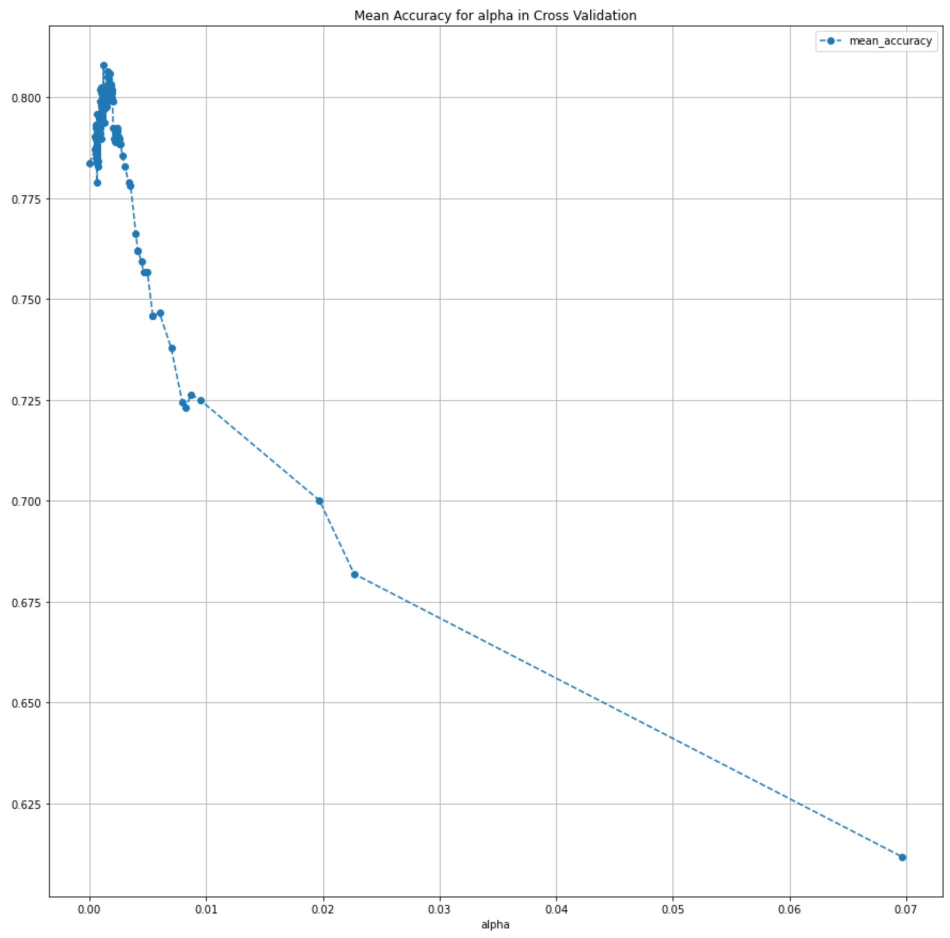
This is done by performing cost complexity pruning which is done by finding the node which is the weakest link and removing a number of these based on a value, alpha, which indicates how much pruning needs to be done with 0 being no pruning and larger values meaning more pruning. Using sklearn's `cost_complexity_pruning_path(X_train, y_train)`, we get a list of appropriate alphas to use in which we can perform cross validation on.

But first, we can get an idea of what values of alpha perform the best by plotting alpha vs the training and testing accuracy of a decision tree trained with that value of alpha.

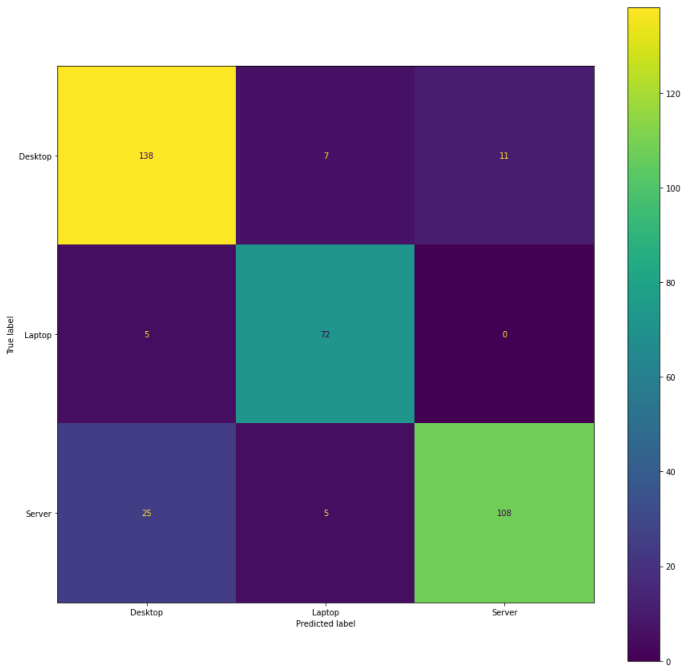


We can see that the data is grossly overfit by looking at the left side of the graph where the gap between training and testing accuracy is significant. We can also see that alpha works well for the training data from alpha around 0.001. We can confirm this alpha by using cross validation.

5-fold cross validation was used and implemented using sklearn. Here is the plot of alpha vs the average cross validation testing accuracy of a decision tree trained with that value of alpha.



Here I choose the alpha that gave the best average testing accuracy which turned out to be alpha = 0.001203. I re-trained the decision tree with this alpha on the whole training dataset and evaluated its performance. This decision tree has a test accuracy of 85.71% (+1.35% increase), a training accuracy of 93.73%, 112 leaves (128 less) and 15 depth (1 less). Its total performance cam be summarized by the following confusion matrix:



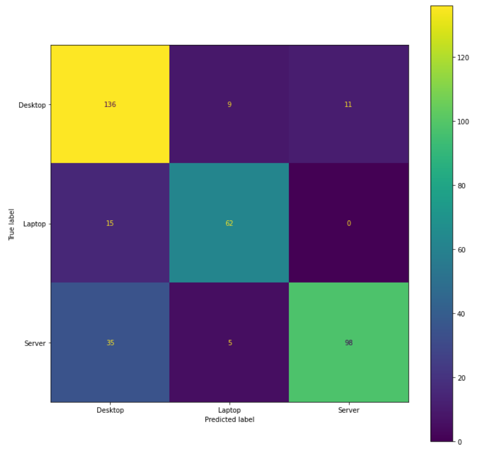
```
            Labels  Test Accuracy
0          Desktop     0.88461538
1           Laptop     0.93506494
2           Server     0.78260870
```

This new decision tree has a better training score with increases in server cpu and desktop cpu classification. However the server classification has decreased by around 4%. This tree, however, is better is it has much better performance in the other classes.

Comparing Classifiers

The previous classifiers were trained with the same random state for the shuffling of the data so that comparisons are fair. Here is the performance

K-Nearest Neighbours Confusion Matrix:

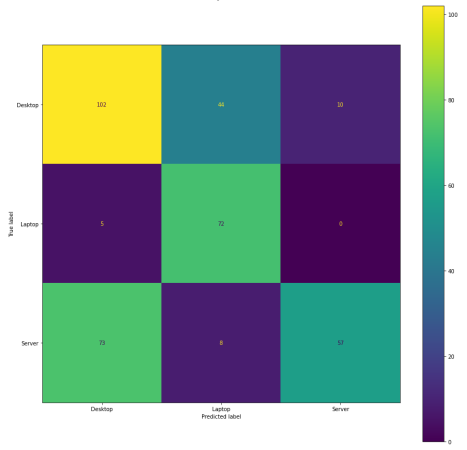


KNN Class Accuracy

```
            Labels  Test Accuracy
0           Desktop    0.87179487
1           Laptop     0.80519481
2           Server     0.71014493
```

The accuracy on the test dataset was 79.78%

Gaussian Naive Bayes Confusion Matrix:

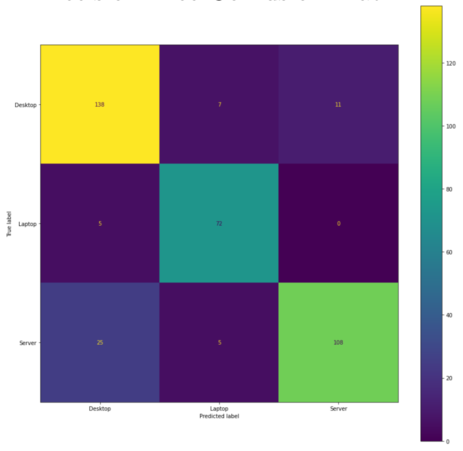


GNB Class Accuracy:

```
            Labels  Test Accuracy
0          Desktop     0.65384615
1           Laptop     0.93506494
2           Server     0.41304348
```

The accuracy on the test dataset was 62.26%

Decision Tree Confusion Matrix:



DT Class Accuracy:

```
         Labels  Test Accuracy
0       Desktop     0.88461538
1        Laptop     0.93506494
2        Server     0.78260870
```

The accuracy on the test dataset was 85.71% The best classifier by far was the decision tree with 85.71% training accuracy.