Heart Disease Prediction

CMPT 353: Computational Data Science

Eshaan Pal Singh | 301416816
Supreet Singh Dhillon | 301417326

SFU

1. Introduction

Cardiovascular diseases (CVDs) are a major cause of death globally. Heart failure is a common event caused by CVDs. In this project we will try to predict if a person has a heart disease or not on the bases of various parameters.

2. Acquiring Relevant Data

While searching for data sets online we found about kaggle and there we found the two most upvoted datasets related to heart conditions which could be found here and here. Thus, we didn't encounter much difficulty in acquiring our data. All it required from our side was some simple google searches. The only thing extra which we did was to extract the heart.csv file from our data archive which Kaggle gave us.

3. Exploratory Data Analysis and Statistics

Once we had our data with us, we straight away started exploring and doing some Data Analysis on it. At first glance, it appeared that our dataset has 918 rows and 12 columns each. These columns were:

- **Age:** Age of the patient (in years)
- **Sex:** Sex of the patient (M or F)
- **Chest Pain Type:** Chest Pain Type (Typical Angina, Atypical Angina, Non-Anginal Pain and Asymptomatic)
- **RestingBP:** Resting Blood Pressure
- **Cholesterol:** Cholesterol level of patient
- **FastingBS:** Fasting Blood Sugar
- **RestingECG:** Resting Electrocardiogram Results (Normal, ST or LVH)
- **MaxHR:** Maximum Heart Rate
- **ExerciseAngina:** Exercise Induced Angina
- **Oldpeak:** Oldpeak
- **ST_Slope:** Slope of Peak Exercise (Up, Down or Flat)
- **HeartDisease:** Patient has heart disease (1) or doesn't (0)

After this, we first and foremost decided to check for any null values. Luckily our dataset didn't have any. Had there been, we would have dropped them instead of taking mean or average from nearby values as doing so might introduce some unwanted bias which can affect our results. Our dataset also didn't have any duplicate values so we didn't have to get rid of any of them.

After that, we inspected the data types of the columns which were given to us. Based on this, we were able to segregate our columns into numerical and categorical values which could be analyzed separately which has been shown below:

- **Numerical Features:** Age, RestingBP, Cholesterol, FastingBS, MaxHR, Oldpeak and HeartDisease
- **Categorical Features:** Sex, ChestPainType, RestingECG, ExerciseAngina, ST_Slope

Based on this, we first analyzed the numerical features and noticed something odd. We described our dataframe and noted the ranges of each of the features which have been added below:

- **Age:** 28 to 77
- **RestingBP:** 0 to 200
- **Cholesterol:** 0 to 603
- **FastingBS:** 0 to 1
- **MaxHR:** 60 to 202
- **Oldpeak:** -2.6 to 6.2

It was here when we first noticed that there is something wrong in our dataset. If one observes carefully, we can see that the minimum value of RestingBP is 0. Even if a person is lying still, having a blood pressure of 0 doesn't seem to be possible and might indicate a fatality (dead person) and so we decided to remove this particular one record from our dataset. We can confirm this behavior with a boxplot too.

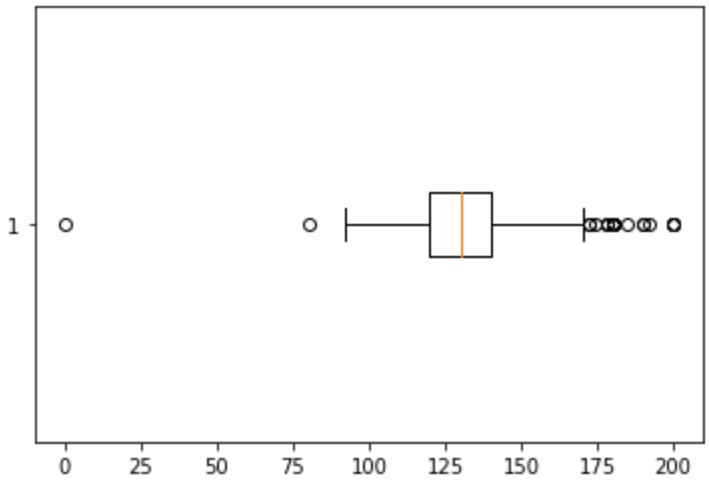


Fig. RestingBP Boxplot with outlier

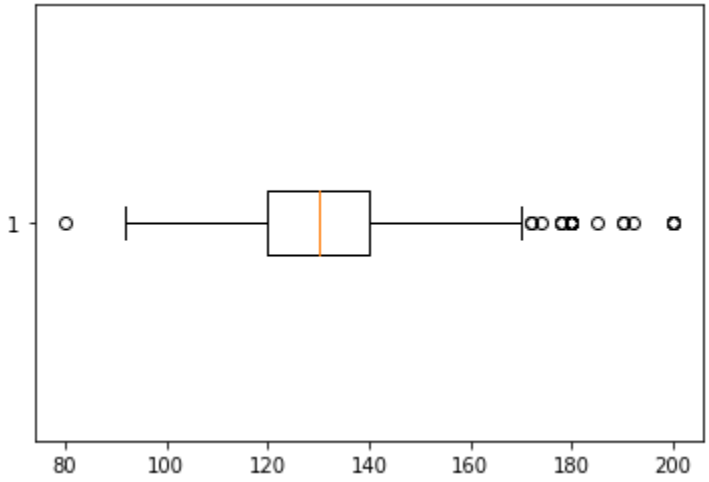


Fig. RestingBP Boxplot with outlier removed

Clearly, even after removing the outlier, there are still some other outliers present. However, we cannot just drop these readings as they are completely valid records irrespective of whether the blood pressure of the patient is towards a higher side or a lower side.

A boxplot of all the quantitative variables can confirm this as shown below.

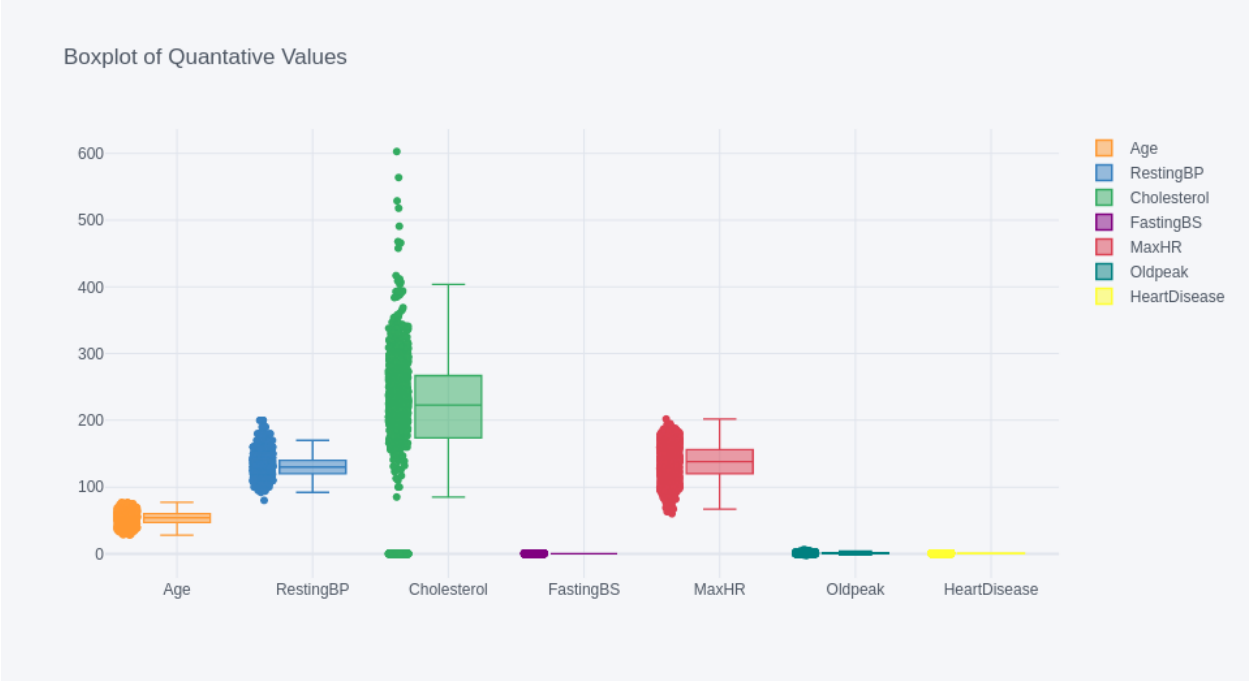


Fig. Boxplot of all numerical variables

Moving further, we decided to check whether any of these features followed a normal distribution or not and so we plotted a histogram of them.

Fig. Histogram of our numerical features

At first glance, it appears that Age, RestingBP, Cholesterol and MaxHR seem to follow normal distribution and we will have to check for Oldpeak values using some sort of statistical test. Also, it also became clear that although FastingBS and HeartDisease were integer values they were still categorical instead of numerical features since their value could only be 0 or 1.

We sampled 50 samples from each variable and performed a normality test on these features and noticed the following p-values:

- Age p-value: 0.450931973189456
- RestingBP p-value: 0.00020574635993458813
- Cholesterol p-value: 0.3706029689788394
- FastingBS p-value: 0.005336336022725512
- MaxHR p-value: 0.5453566161673747
- Oldpeak p-value: 8.949422050432128e-06
- HeartDisease p-value: 0.0

Thus, based on the above tests it can be safely assumed that Age, RestingBS, Cholesterol, MaxHR and Oldpeak values come from a normal distribution (since p-value > 0.05) whereas FastingBS and HeartDisease do not follow a Normal Distribution (as we proved earlier since they were categorical variables) and so we will be moving them to our list of categorical values. (p-value < 0.05)

Moving further ahead to our categorical variables, we first plot them.

Fig. Histogram of Categorical variables

Next, we try to find if any of these variables are correlated to each other and so we use another test from our data science toolbox - The Chi Square Test. We constructed contingency tables of all these variables against HeartDisease and then calculated the chi square p-value. Turns out all of the variables are related to HeartDisease as is evident from the values below and thus we'll be using all of them (since p-value < 0.05)

- Sex and HeartDisease are correlated to each other. P-value: 5.2809560649710597e-20
- ChestPainType and HeartDisease are correlated to each other. P-value: 5.3488219185881074e-58
- RestingECG and HeartDisease are correlated to each other. P-value: 0.003947123023954998
- ExerciseAngina and HeartDisease are correlated to each other. P-value: 1.899519261220225e-50
- ST_Slope and HeartDisease are correlated to each other. P-value: 7.566932580265743e-78
- FastingBS and HeartDisease are correlated to each other. P-value: 9.215944248776643e-16

4. Machine Learning

Now that we have some slight info about what and how our data looks like, we further proceed with our Machine Learning part. It is evident by now that our problem is a classification problem and not a regression one and so we will be using only our classification algorithms.

- First, we convert our ordinal features to quantitative features (categorical -> numerical)
- After that, we split our dataset into train and valid sets. For this, we keep 15% of the data aside as our validation test.
- Scale our values to make them all fit in a similar range.
- We used 7 different models for classification. For each model, we first trained a base model and then tried to find the optimal hyperparameters using GridSearchCV.

1) Gaussian Naive Bayes:

Gaussian Naive Bayes didn't have much to optimize except for a var_smoothing parameter and thus the base model and the tuned models were almost similar in performance to each other.

```
Classifier :  GaussianNB
Train Score:  0.863
Valid Score:  0.862

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.87      0.85        62
           1       0.89      0.86      0.87        76

    accuracy                           0.86       138
   macro avg       0.86      0.86      0.86       138
weighted avg       0.86      0.86      0.86       138
```

Confusion Matrix

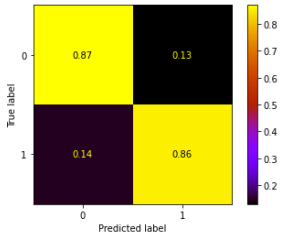


Fig. Base GaussianNB Model

```
Classifier :  GaussianNB
Train Score:  0.867
Valid Score:  0.870

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.89      0.86        62
           1       0.90      0.86      0.88        76

    accuracy                           0.87       138
   macro avg       0.87      0.87      0.87       138
weighted avg       0.87      0.87      0.87       138
```

Confusion Matrix

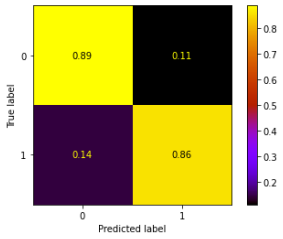


Fig. GaussianNB after tuning

We were able to find out if we set 'var_smoothing' to 1.0 we're able to maximize our model performance. Although there is some improvement, it is almost negligible.

2) SVC:

```
Classifier :  SVC
Train Score:  0.918
Valid Score:  0.870

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.84      0.85        62
           1       0.87      0.89      0.88        76

    accuracy                           0.87       138
   macro avg       0.87      0.87      0.87       138
weighted avg       0.87      0.87      0.87       138
```

Confusion Matrix

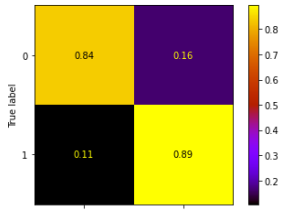


Fig. Base SVC Model

In terms of SVC, we have three parameters to tune. On performing a grid search, we find that setting C=1, gamma=0.1 and kernel='rbf' gives us the most optimal value although the improvement in performance is still almost negligible.

```
Train Score:  0.924
Valid Score:  0.891

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.87      0.88        62
           1       0.90      0.91      0.90        76

    accuracy                           0.89       138
   macro avg       0.89      0.89      0.89       138
weighted avg       0.89      0.89      0.89       138
```

Confusion Matrix

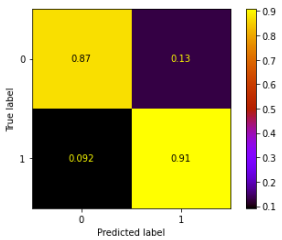


Fig. SVC Tuned Model

3) KNearestNeighbors:

```
Classifier :  KNN
Train Score:  0.900
Valid Score:  0.884

Classification Report:
              precision    recall  f1-score   support

           0       0.90      0.84      0.87        62
           1       0.88      0.92      0.90        76

    accuracy                           0.88       138
   macro avg       0.89      0.88      0.88       138
weighted avg       0.88      0.88      0.88       138
```

Confusion Matrix

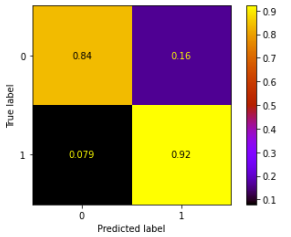


Fig. KNN Base Model

Tuning a KNN model is relatively easier as there is only one hyperparameter to tune, the number of nearest neighbors. On performing a grid search, the optimal value came out to be n_neighbors = 17. However, this time we noted a slight decrease in accuracy but overall better results in the confusion matrix.

```
Classifier :  KNN
Train Score:  0.882
Valid Score:  0.877

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.87      0.86        62
           1       0.89      0.88      0.89        76

    accuracy                           0.88       138
   macro avg       0.88      0.88      0.88       138
weighted avg       0.88      0.88      0.88       138
```

Confusion Matrix

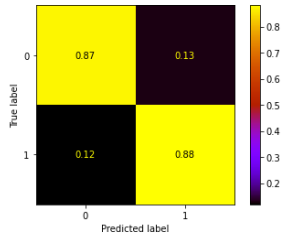


Fig. KNN Tuned Model

4) Decision Trees:

```
Classifier :  Decision Trees
Train Score:  1.000
Valid Score:  0.790

Classification Report:
              precision    recall  f1-score   support

           0       0.76      0.77      0.77        62
           1       0.81      0.80      0.81        76

    accuracy                           0.79       138
   macro avg       0.79      0.79      0.79       138
weighted avg       0.79      0.79      0.79       138
```

Confusion Matrix

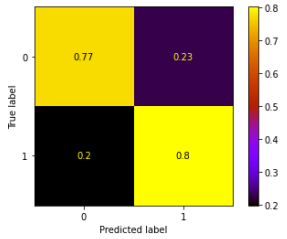


Fig. Base Decision Trees Model

In terms of decision trees, we optimized four hyperparameters. We set their values as 'criterion' = 'gini', 'max_depth' = 5, 'min_samples_leaf' = 2, 'min_samples_split' = 8.

It is to be noted that although our accuracy decreased from before doing this, at least our model is no longer overfitting as before.

```
Classifier :  Decision Trees
Train Score:  0.896
Valid Score:  0.855

Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.87      0.84        62
           1       0.89      0.84      0.86        76

    accuracy                           0.86       138
   macro avg       0.85      0.86      0.85       138
weighted avg       0.86      0.86      0.86       138
```

Confusion Matrix

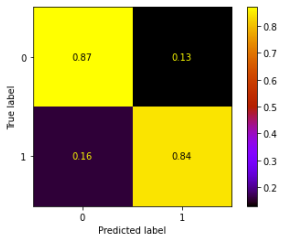


Fig. Tuned Decision Tree Model

5) Random Forest:

```
Classifier :  Random Forest
Train Score:  1.000
Valid Score:  0.870

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.85      0.85        62
           1       0.88      0.88      0.88        76

    accuracy                           0.87       138
   macro avg       0.87      0.87      0.87       138
weighted avg       0.87      0.87      0.87       138


Confusion Matrix
```

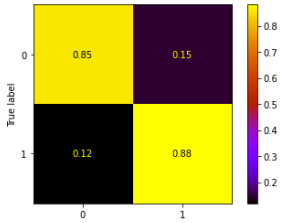


Fig. Base Random Forest Model

Again, like before we try to find the optimal hyperparameters and came up with these hyperparameters upon using GridSearchCV.

{'criterion': 'entropy', 'max_depth': 7, 'max_features': 'auto', 'n_estimators': 100}

Again, although the entropy decreased from before, at least our model does not overfit anymore.

```
Classifier :  Random Forest
Train Score:  0.935
Valid Score:  0.877

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.84      0.86        62
           1       0.87      0.91      0.89        76

    accuracy                           0.88       138
   macro avg       0.88      0.87      0.87       138
weighted avg       0.88      0.88      0.88       138


Confusion Matrix
```

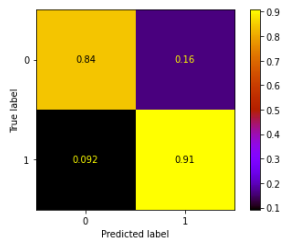


Fig. Tuned Random Forest Model

6) Gradient Boosting:

```
Classifier :  Gradient Boosting
Train Score:  0.946
Valid Score:  0.870

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.87      0.86        62
           1       0.89      0.87      0.88        76

    accuracy                           0.87       138
   macro avg       0.87      0.87      0.87       138
weighted avg       0.87      0.87      0.87       138
```

Confusion Matrix

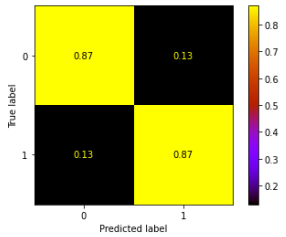


Fig. Base Gradient Boosting

```
Classifier :  Gradient Boosting
Train Score:  0.838
Valid Score:  0.848

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.76      0.82        62
           1       0.82      0.92      0.87        76

    accuracy                           0.85       138
   macro avg       0.86      0.84      0.84       138
weighted avg       0.85      0.85      0.85       138
```

Confusion Matrix

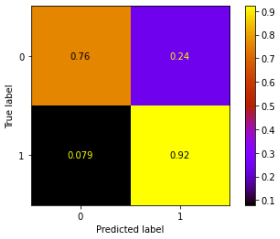
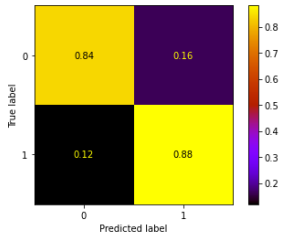


Fig. Tuned Gradient Boosting

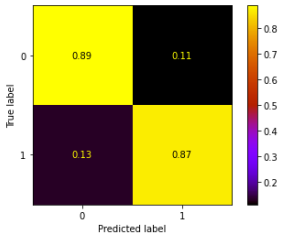
7) MLP Classifier:

```
Classifier :  MLP Classifier
Train Score:  0.926
Valid Score:  0.862

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.84      0.85        62
           1       0.87      0.88      0.88        76

    accuracy                           0.86       138
   macro avg       0.86      0.86      0.86       138
weighted avg       0.86      0.86      0.86       138


Confusion Matrix
```

```
/home/ubuntu/project/lib/python3.8/site-packages/sklearn/
  warnings.warn(
```



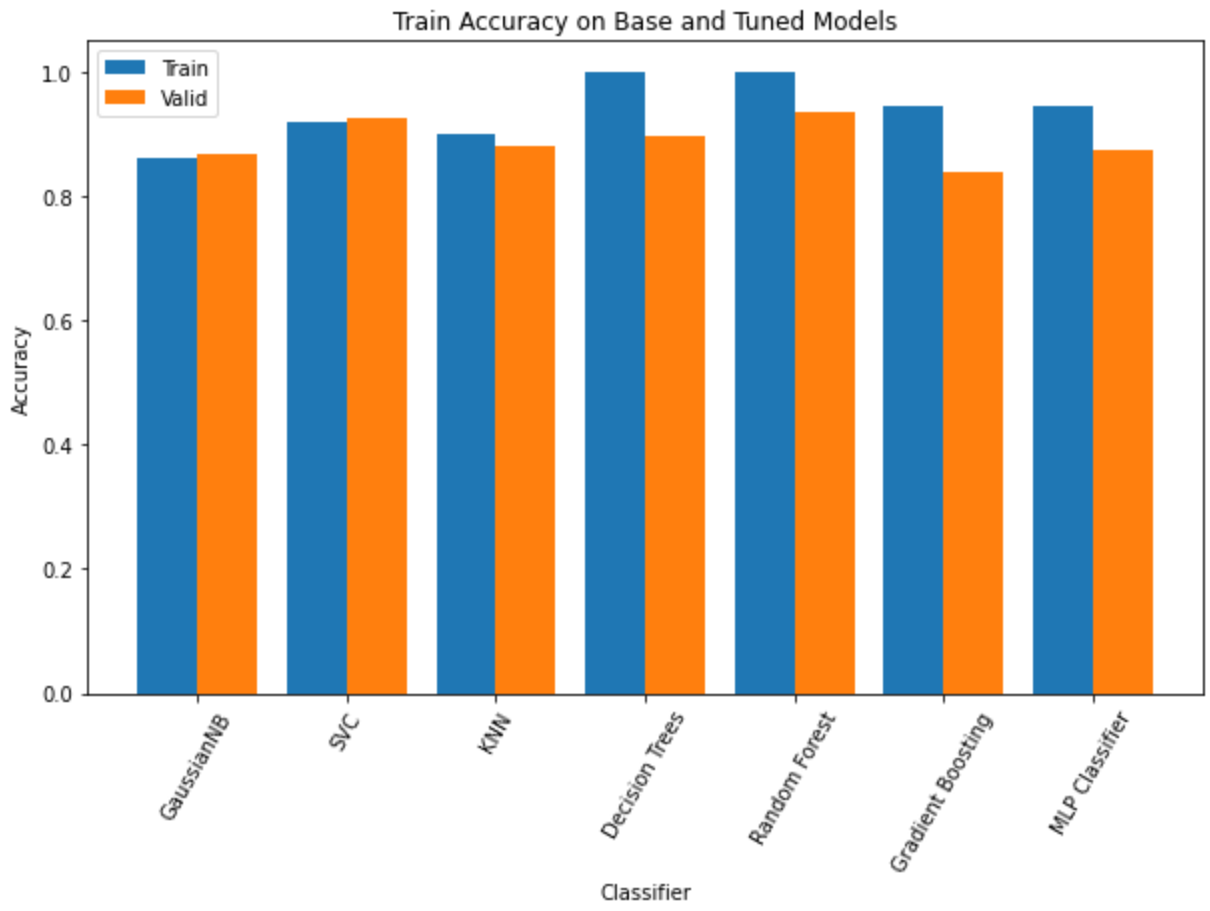

```
Classifier :  MLP Classifier
Train Score:  0.876
Valid Score:  0.877

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.89      0.87        62
           1       0.90      0.87      0.89        76

    accuracy                           0.88       138
   macro avg       0.88      0.88      0.88       138
weighted avg       0.88      0.88      0.88       138


Confusion Matrix
```



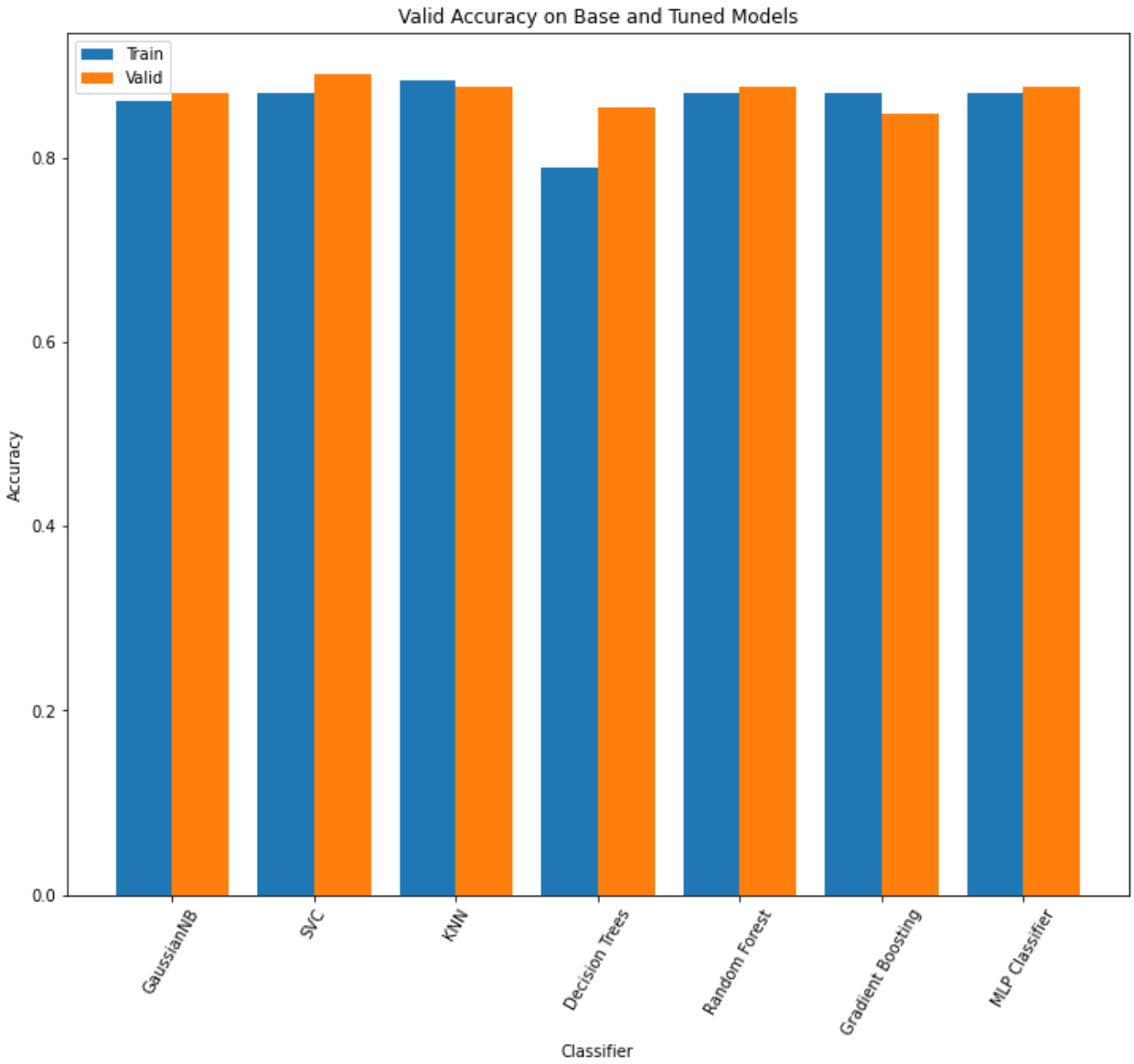
It is to be noted that although our accuracy might have decreased while performing hyperparameter tuning on Gradient Boosting and MLP Classifier, it is because we weren't able to run full Grid Search due to lack of resources and can only run on some variations of the input parameters.

5. Results

First, we plot the train accuracy of our base and tuned models.



It appears that our Decision Trees and Random Forest seem to perform the best on train accuracy, However, this can be deceiving as this is our train accuracy and might be overfitted. So, we have a look at our valid accuracy



Thus, looking at this, it appears that almost all of our models are at par with each other. However, the tuned SVC seems to perform the best out of all.

6. Limitations

Since it was our groups first time working on a machine learning project like this we did not know it would take us so much time to fine tune our Gradient Boosting and MLP models and so we couldn't tune them properly. If we had enough resources and time, we feel that they could have easily outperformed the others.