Machine Learning Assignment

Regression Analysis Report

Name: Omkesh Rane

1. Problem Statement

An Individual has started his own mobile company. He wants to give a tough fight to big companies like Apple, Samsung, etc.

He does not know how to estimate the price of mobiles his company creates. In this competitive mobile phone market, you cannot simply assume things. To solve this problem he collects sales data of mobile phones of various companies.

Bob wants to find out some relation between the features of a mobile phone (eg:- RAM, Internal Memory, etc) and its selling price. But he is not so good at Machine Learning. So he needs your help to solve this problem.

In this problem, you do not have to predict the actual price but a price range indicating how high the price is:

- **Data Set**

battery Power: Total energy a battery can store in one time measured in mAh.

clock_speed: speed at which microprocessor executes instructions.

dual Sim: Has dual sim support or not

fc: Front camera in megapixel

four_g: Has 4g support or not

int_memory: Internal memory in gigabyte

m_depth: Mobile depth in cm

mobile_wt: Weight of the mobile phone

n_cores: Number of cores

pc: Primary camera megapixel

px_height: Pixel resolution height

px_width: Pixel resolution width

ram: Random access memory in gigabytes

sc_h: Screen height of mobile in cm

sc_w: Screen width of mobile in cm

talk_time: longest time that a single battery charge will last when you are

three_g: Has 3G or not

price_range(Target variable): price range of the mobile.

Our Target variable here would be the price_range

- Given the above dataset here we have to apply the liner regression models to find the relation between the components and the price estimation of the mobile device.

- So basis on the given dataset we would first implement the Linear Regression model without regularization.

- From the given data we have total 18 independent input variable and out of that one is our target variable which is **price_range.**

1. Implementation of Linear Regression without regression.

- From the above data we can see that the data is interpreted clearly and we don't have to add an extra step in the code for categorial variables into numerical variables.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
1	battery_p	clock_spe	dual_sim	fc	four_g	int_memc	m_dep	mobile_w	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	price_range	
2	842	2.2	No	1	No	7	0.6	188	2	2	20	756	2549	9	7	19	No	284	
3	1021	0.5	Yes	0	Yes	53	0.7	136	3	6	905	1988	2631	17	3	7	Yes	653	
4	563	0.5	Yes	2	Yes	41	0.9	145	5	6	1263	1716	2603	11	2	9	Yes	679	
5	615	2.5	No	0	No	10	0.8	131	6	9	1216	1786	2769	16	8	11	Yes	484	
6	1821	1.2	No	13	Yes	44	0.6	141	2	14	1208	1212	1411	8	2	15	Yes	387	
7	1859	0.5	Yes	3	No	22	0.7	164	1	7	1004	1654	1067	17	1	10	Yes	298	
8	1821	1.7	No	4	Yes	10	0.8	139	8	10	381	1018	3220	13	8	18	Yes	1262	
9	1954	0.5	Yes	0	No	24	0.8	187	4	0	512	1149	700	16	3	5	Yes	161	
10	1445	0.5	No	0	No	53	0.7	174	7	14	386	836	1099	17	1	20	Yes	178	
11	509	0.6	Yes	2	Yes	9	0.1	93	5	15	1137	1224	513	19	10	12	Yes	158	
12	769	2.9	Yes	0	No	9	0.1	182	5	1	248	874	3946	5	2	7	No	818	
13	1520	2.2	No	5	Yes	33	0.5	177	8	18	151	1005	3826	14	9	13	Yes	894	
14	1815	2.8	No	2	No	33	0.6	159	4	17	607	748	1482	18	0	2	Yes	377	
15	803	2.1	No	7	No	17	1	198	4	11	344	1440	2680	7	1	4	Yes	529	
16	1866	0.5	No	13	Yes	52	0.7	185	1	17	356	563	373	14	9	3	Yes	201	
17	775	1	No	3	No	46	0.7	159	2	16	862	1864	568	17	15	11	Yes	201	
18	838	0.5	No	1	Yes	13	0.1	196	8	4	984	1850	3554	10	9	19	Yes	1184	
19	595	0.9	Yes	7	Yes	23	0.1	121	3	17	441	810	3752	10	2	18	Yes	880	
20	1131	0.5	Yes	11	No	49	0.6	101	5	18	658	878	1835	19	13	16	Yes	378	
21	682	0.5	No	4	No	19	1	121	4	11	902	1064	2337	11	1	18	No	272	
22	772	1.1	Yes	12	No	39	0.8	81	7	14	1314	1854	2819	17	15	3	Yes	1391	
23	1709	2.1	No	1	No	13	1	156	2	2	974	1385	3283	17	1	15	Yes	1258	
24	1949	2.6	Yes	4	No	47	0.3	199	4	7	407	822	1433	11	5	20	No	349	
25	1602	2.8	Yes	4	Yes	38	0.7	114	3	20	466	788	1037	8	7	20	Yes	198	
26	503	1.2	Yes	5	Yes	8	0.4	111	3	13	201	1245	2583	11	0	12	Yes	366	
27	961	1.4	Yes	0	Yes	57	0.6	114	8	3	291	1434	2782	18	9	7	Yes	499	
28	519	1.6	Yes	7	Yes	51	0.3	132	4	19	550	645	3763	16	1	4	Yes	1265	
29	956	0.5	No	1	Yes	41	1	143	7	6	511	1075	3286	17	8	12	Yes	1218	
30	1453	1.6	Yes	12	Yes	52	0.3	96	2	18	187	1311	2373	10	1	10	Yes	456	
31	851	0.5	No	3	No	21	0.4	200	5	7	1171	1263	478	12	7	10	Yes	125	
32	1579	0.5	Yes	0	No	5	0.2	88	7	9	1358	1739	3532	17	11	12	No	887	
33	1568	0.5	No	16	No	33	1	150	8	20	413	654	508	5	1	6	Yes	131	
34	1319	0.9	No	3	Yes	41	0.9	107	1	18	85	1152	2227	18	5	3	Yes	374	
35	1310	2.2	Yes	0	Yes	51	0.6	100	4	0	178	1919	3845	7	0	12	Yes	1435	
36	644	2.7	No	0	No	22	0.7	157	8	3	311	881	1262	12	1	15	Yes	227	
37	725	1.3	Yes	16	No	60	0.4	160	8	17	1134	1249	1326	10	4	15	Yes	437	
38	589	2.3	Yes	1	No	61	0.6	160	4	10	429	815	2113	13	7	2	Yes	268	

- As the categorial input variable which include **dual_sim, fc,four_g and three_g** are already in the form of numerical variables that are 1’s and 0’s.

- so will directly import the data set in the code and proceed with the initial data analysis step.

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import SGDRegressor
import plotly.graph_objs as go
import plotly.figure_factory as ff

# Importing dataset and examining it
dataset = pd.read_csv("C:\Program Files\JetBrains\PyCharm Community Edition 2022.2.2\Bin\Read\Mobi.csv")
pd.set_option('display.max_columns', None) # to make sure you can see all the columns in output window
print(dataset.head())
print(dataset.shape)
print(dataset.info())
print(dataset.describe())


print(dataset.info())

# Plotting Correlation Heatmap
corrs = dataset.corr()
figure = ff.create_annotated_heatmap(
    z=corrs.values,
    x=list(corrs.columns),
    y=list(corrs.index),
    annotation_text=corrs.round(2).values,
    showscale=True)
figure.show()

```

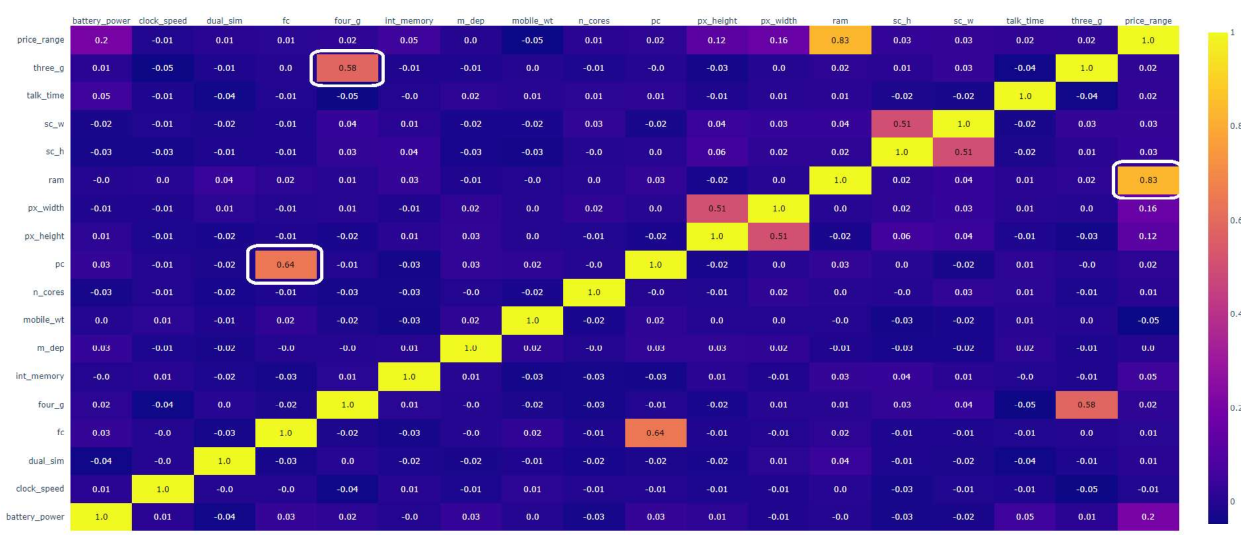
- Basis on the above figure we would be also using the plotly package so we can find out the corelation and causation in our data set.

- Running the above code we get the following output.

```
RangeIndex: 2000 entries, 0 to 1999
Data columns (total 18 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   battery_power  2000 non-null   int64
 1   clock_speed    2000 non-null   float64
 2   dual_sim       2000 non-null   int64
 3   fc             2000 non-null   int64
 4   four_g         2000 non-null   int64
 5   int_memory     2000 non-null   int64
 6   m_dep          2000 non-null   float64
 7   mobile_wt      2000 non-null   int64
 8   n_cores        2000 non-null   int64
 9   pc             2000 non-null   int64
 10  px_height      2000 non-null   int64
 11  px_width       2000 non-null   int64
 12  ram            2000 non-null   int64
 13  sc_h           2000 non-null   int64
 14  sc_w           2000 non-null   int64
 15  talk_time      2000 non-null   int64
 16  three_g        2000 non-null   int64
 17  price_range    2000 non-null   int64
dtypes: float64(2), int64(16)
```

- We can assume from the above result that there are no null values in our data and all the variables are of int type, this indicates that our data set is proper and can be used for further implementation.

- We also get the correlation heatmap which is plotted with the help of plotly package.



- After analyzing the heatmap we can see that there is a high correlation between we can say that there is high positive correlation 0.83 between the **ram** and **price range** which is making sense because increasing the Ram component in the mobile will increase the selling price on the mobile.

- As per the domain interpretation we can also say that there is causation here because the increase the one value here is increasing the other value.

- As the price_range is our target variable we would be already dropping it.

- The Second correlation we are getting here is between three_g and four_g but we are not sure about this because 3g and 4g are the communication band and they are totally independent, because 3g works on different technical sets and 4g works on different technical set, so we won't be dropping either of them.

- And with respect to domain the cost of a 4g mobile will always be higher than a 3g mobile.

- The third correlation we got here is between pc(primary camera) and fc(front camera) we can also say that there is causation here because mobiles with higher megapixels of front camera will have higher megapixels in primary camera.

- So in this case we would be dropping either of the ones, we will drop pc(primary camera variable here.

- So from the interoperation we would be dropping two variables which will include price_range our target variable and pc(primary camera)

```
# Dividing dataset into label and feature sets
X = dataset.drop(['price_range', 'pc'], axis = 1) # Features
Y = dataset['price_range'] # Labels
print(type(X))
print(type(Y))
print(X.shape)
print(Y.shape)
```

- We would further move on to tune our liner regression algorithm.

```python
sgdr = SGDRegressor(random_state = 1, penalty = None)
grid_param = {'eta0': [.0001, .001, .01, .1, 1], 'max_iter':[10000, 20000, 30000, 40000]}

gd_sr = GridSearchCV(estimator=sgdr, param_grid=grid_param, scoring='r2', cv=5)

gd_sr.fit(X_scaled, Y)

best_parameters = gd_sr.best_params_
print("Best parameters: ", best_parameters)

best_result = gd_sr.best_score_ # Mean cross-validated score of the best_estimator
print("r2: ", best_result)

Adj_r2 = 1-(1-best_result)*(1600-1)/(1600-16-1)
print("Adjusted r2: ", Adj_r2)
```

- As we are only implementing linear regression without regularization here so our penalty parameter would be none here.

- So in here we would only tune two hyperparameters which is “eta0” which is a learning rate parameter and our values would be between [.0001,.001,.01,.1,1] as it would control the step size of stochastic gradient descent.

- Our second hyperparameter is max_iter which will define the maximum iteration of the stochastic gradient descent our value will include [1000,20000,30000,40000], if we don’t define the value here then our algorithm will keep on running and won’t stop.

- We are using grid search cv to tune both of the parameters our scoring parameter or evaluation matrix will be r2 which will give us the sum of square prediction errors which will give us the percentage of the variance at our final output.

- Our CV (k-fold cross-validation) will be 5.

- To overcome the misleading values of r2 we would be using another parameter which is adjusted r2 which goes by the formula 1-(1-best_result)*(n-1)/(n-p-1).

- Where in n is the number of rows in our data set divided by our Cv which is 5 and multiplied by 4.

- P indicates the number of independent variables in our dataset, which will include 16 as we have dropped two from the total variable which was 18

- So if we do the math we have 2000 rows of 2000/5 which is 400 so here one fold will have 400 rows and after multiplying it by 4 we get the value 1600.

- So our adjusted r2 value would be **1-(1-best_result)*(1600-1)/(1600-16-1).**

- After running the code with the above tuning we get the below output.

```
Best parameters:  {'eta0': 0.01, 'max_iter': 10000}
r2:  0.7576509600822158
Adjusted r2:  0.7552014435700967
Intercept:  [547.72493244]
         Features  Coefficients
11            ram    303.686925
0   battery_power     73.187875
10       px_width     44.324383
9       px_height     29.644485
5      int_memory      8.480293
12           sc_h      6.901240
4          four_g      2.494717
15        three_g      2.019998
8         n_cores      1.602791
6           m_dep      1.001385
14      talk_time      0.963809
3              fc     -3.111704
1     clock_speed     -5.869576
13           sc_w     -7.033832
2        dual_sim     -7.657079
7       mobile_wt    -14.849966
```

- Our r2 value is 0.755201 which means 75 percent of mobile price estimation is explained by our 16 independent variables.

- This model was created using a small step size as eta0 is 0.01 so just 1 pe was used to converge to minima this is our optimal step size

- We are converging in under 10000 steps as this is quite a small dataset.

- The adjusted score is somewhat the same because all the variables used here are meaningful variables.

- With the above output we can conclude that Ram is the major deciding factor in estimating the mobile price followed by battery power, we can say that if ram size is increased by a certain value then it will add up in the mobile selling value providing other variable has no change.

- Same goes for the battery power if battery power is increased by a certain mah it will add up in the mobile price.

- We cannot say that this is a proper interpretation as we are not adding overfitting avoidance mechanism which is regularization.

1. Implementation of Linear Regression with regularisation.

1.1 Elastic net

- We will use the same dataset with same tuning and apply elastic net regularisation.

```python
# Dividing dataset into label and feature sets
X = dataset.drop(['price_range', 'pc'], axis = 1) # Features
Y = dataset['price_range'] # Labels
print(type(X))
print(type(Y))
print(X.shape)
print(Y.shape)
```

```python
# Linear Regression with Regularization
# Tuning the SGDRegressor parameters 'eta0' (learning rate) and 'max_iter', along with the regularization parameter alpha using Grid Search
sgdr = SGDRegressor(random_state = 1, penalty = 'elasticnet')
grid_param = {'eta0': [.0001, .001, .01, .1, 1], 'max_iter':[10000, 20000, 30000, 40000],'alpha': [.001, .01, .1, 1,10, 100],'l1_ratio':[0.25,0.5,0.75]}


gd_sr = GridSearchCV(estimator=sgdr, param_grid=grid_param, scoring='r2', cv=5)

gd_sr.fit(X_scaled, Y)

best_parameters = gd_sr.best_params_
print("Best parameters: ", best_parameters)

best_result = gd_sr.best_score_ # Mean cross-validated score of the best_estimator
print("r2: ", best_result)

Adj_r2 = 1-(1-best_result)*(1600-1)/(1600-16-1)
print("Adjusted r2: ", Adj_r2)
```

- Here in we will add two more hyperparameters which would be alpha and l1_ratio.

- Wherein l1_ration will define the mix of two in the elastic net.

- Alpha will control how much your coefficient will shrink the larger its value more the co will shrink ,less the value less the coefficient will shrink.

- The output of the above result will be as follows

```
Best parameters:  {'alpha': 0.01, 'eta0': 0.01, 'l1_ratio': 0.75, 'max_iter': 10000}
r2:  0.7576806283276252
Adjusted r2:  0.7552314116840636
Intercept:  [547.73614771]
         Features  Coefficients
11            ram    302.911653
0   battery_power     72.993551
10       px_width     44.230413
9       px_height     29.618885
5      int_memory      8.491924
12           sc_h      6.848390
4          four_g      2.490872
15        three_g      1.947934
8         n_cores      1.586590
14      talk_time      0.927132
6           m_dep      0.833953
3              fc     -3.078598
1     clock_speed     -5.845780
13           sc_w     -6.947379
2        dual_sim     -7.607966
7       mobile_wt    -14.788870
```

- As we can see from the above result we are getting the same performance which was expected because we don’t have any less meaningful variables in our dataset.

- Our alpha value is 0.01 which is the smallest value which tells us that lesser variable has been shrieked

- Our l1(lasso) ratio mix here is 0.75 which means 75 percent l1 is used here and 25 l2(rigid) is used.

1.2 L1(lasso)

- Implementing l1 lasso regularization on the same dataset.

```python
# Linear Regression with Regularization
# Tuning the SGDRegressor parameters 'eta0' (learning rate) and 'max_iter', along with the regularization parameter alpha using Grid Search
sgdr = SGDRegressor(random_state = 1, penalty = 'elasticnet')
grid_param = {'eta0': [.0001, .001, .01, .1, 1], 'max_iter':[10000, 20000, 30000, 40000],'alpha': [.001, .01, .1, 1,10, 100],'l1_ratio':[1]}


gd_sr = GridSearchCV(estimator=sgdr, param_grid=grid_param, scoring='r2', cv=5)

gd_sr.fit(X_scaled, Y)

best_parameters = gd_sr.best_params_
print("Best parameters: ", best_parameters)

best_result = gd_sr.best_score_ # Mean cross-validated score of the best_estimator
print("r2: ", best_result)

Adj_r2 = 1-(1-best_result)*(1600-1)/(1600-16-1)
print("Adjusted r2: ", Adj_r2)
```

- We get the following output.

```
Best parameters:  {'alpha': 1, 'eta0': 0.0001, 'l1_ratio': 1, 'max_iter': 10000}
r2:  0.7580360597943211
Adjusted r2:  0.7555904356355776
Intercept:  [547.76326673]
         Features  Coefficients
11            ram    305.756639
0   battery_power     72.097615
10       px_width     43.053739
9       px_height     27.566616
5      int_memory      7.773898
12           sc_h      5.243113
8         n_cores      2.883396
4          four_g      1.837763
14      talk_time      0.895426
15        three_g      0.792475
6           m_dep      0.000000
3              fc     -1.389110
13           sc_w     -1.838580
1     clock_speed     -2.400814
2        dual_sim     -4.616369
7       mobile_wt    -15.566781
```

- As we can see from the above result we are getting the same performance which was expected because we don't have any less meaningful variables in our dataset.

- Our alpha value is 1 which is the largest value that tells us that the maximum variable has been shrieked.

- Here in this less significant variable has been shrinked to 0 , in our case it is m_depth(mobile depth in cm)

- But still its giving the same performance.

1.3 L2(Rigid):

- Implementing l2 Rigid regularization on the same dataset.

```python
sgdr = SGDRegressor(random_state = 1, penalty = 'elasticnet')
grid_param = {'eta0': [.0001, .001, .01, .1, 1], 'max_iter':[10000, 20000, 30000, 40000],'alpha': [.001, .01, .1, 1,10, 100],'l1_ratio':[0]}


gd_sr = GridSearchCV(estimator=sgdr, param_grid=grid_param, scoring='r2', cv=5)

gd_sr.fit(X_scaled, Y)

best_parameters = gd_sr.best_params_
print("Best parameters: ", best_parameters)

best_result = gd_sr.best_score_ # Mean cross-validated score of the best_estimator
print("r2: ", best_result)

Adj_r2 = 1-(1-best_result)*(1600-1)/(1600-16-1)
print("Adjusted r2: ", Adj_r2)
```

- We get the following output

```
Best parameters:  {'alpha': 0.001, 'eta0': 0.01, 'l1_ratio': 0, 'max_iter': 10000}
r2:  0.7576514168942385
Adjusted r2:  0.7552019049992972
Intercept:  [547.72656768]
         Features  Coefficients
11            ram    303.380708
0   battery_power     73.113280
10       px_width     44.290246
9       px_height     29.634721
5      int_memory      8.485273
12           sc_h      6.885204
4          four_g      2.493562
15        three_g      2.026207
8         n_cores      1.599587
6           m_dep      1.003479
14      talk_time      0.972484
3              fc     -3.105440
1     clock_speed     -5.865473
13           sc_w     -7.009264
2        dual_sim     -7.640892
7       mobile_wt    -14.831622
```

- As we can see from the above result we are getting the same performance that was expected because we don't have any less meaningful variables in our dataset.

- Our alpha value is 0.001 which is the lowest value that tells us that the lesser variable has been shrieked.

- So we can conclude here that by implementing regression without regularisation and with regularisation which includes elasticnet, L1 and L2 we are getting the same performance there is not major change in to r2 score and adjusted r2 score so we basis on that we can interpret that our model is neither overfitting nor underfitting.

2. Support Vector Regression

- Implementation of SVC

```
# Dividing dataset into label and feature sets
X = dataset.drop(['price_range','pc'], axis = 1) # Features
Y = dataset['price_range'] # Labels
print(type(X))
print(type(Y))
print(X.shape)
print(Y.shape)

# Normalizing numerical features so that each feature has mean 0 and variance 1
feature_scaler = StandardScaler()
X_scaled = feature_scaler.fit_transform(X)

##########################################################################################
# Implementing Support Vector Regression
# Tuning the SVR parameters 'kernel', 'C', 'epsilon' and implementing cross-validation using Grid Search
svr = SVR()
grid_param = {'kernel': ['linear', 'poly', 'rbf', 'sigmoid'], 'C': [100,1000,10000],'epsilon':[100,1000,10000]}

gd_sr = GridSearchCV(estimator=svr, param_grid=grid_param, scoring='r2', cv=5)

gd_sr.fit(X_scaled, Y)

best_parameters = gd_sr.best_params_
print("Best parameters: ", best_parameters)

best_result = gd_sr.best_score_ # Mean cross-validated score of the best_estimator
print("r2: ", best_result)

Adj_r2 = 1-(1-best_result)*(1600-1)/(1600-16-1)
print("Adjusted r2: ", Adj_r2)
```

- In SVC we will tune following hyperparameter kernel, C and Epsilon.

- We get the following output:

```
Best parameters:  {'C': 1000, 'epsilon': 100, 'kernel': 'linear'}
r2:  0.7512778597702459
Adjusted r2:  0.7487639278412022

Process finished with exit code 0
```

- In the above output we can see that the value of C is larger which means lesser number of variables has been used but our kernel is linear which means our model is same as linear regression model.
- In this also we get the same performance which we got in the previous regression models

3. Random Forest regressor:

- Implementation of Random Frost Regressor.

```
rfr = RandomForestRegressor(criterion='squared_error', max_features='sqrt', random_state=1)
grid_param = {'n_estimators': [10,20,30,50,100]}

gd_sr = GridSearchCV(estimator=rfr, param_grid=grid_param, scoring='r2', cv=5)

gd_sr.fit(X_scaled, Y)

best_parameters = gd_sr.best_params_
print("Best parameters: ", best_parameters)

best_result = gd_sr.best_score_# Mean cross-validated score of the best_estimator
print("r2: ", best_result)

Adj_r2 = 1-(1-best_result)*(1600-1)/(1600-16-1)
print("Adjusted r2: ", Adj_r2)
```

- In this we tune the same hyperparameter which we tuned for random forest classification, we get the following output:

```
Best parameters:  {'n_estimators': 100}
r2:  0.7784498789073337
Adjusted r2:  0.7762105852007749
ram              0.633992
battery_power    0.071294
px_width         0.051444
px_height        0.043855
mobile_wt        0.030808
int_memory       0.025769
sc_w             0.021753
talk_time        0.020762
clock_speed      0.020430
sc_h             0.019609
fc               0.017531
m_dep            0.016136
n_cores          0.015305
dual_sim         0.004433
four_g           0.003641
three_g          0.003238
dtype: float64
```

- We can see that the random forest regressor is giving us better performance than the other three models and this states that there is some non-linearity in the data.

- Basis on the feat map we retune the model and add most significant variable which are there in the feat map and increasing the count of n_Estimators.

```python
X_ = dataset[['ram','battery_power','px_width','px_height','mobile_wt','int_memory','pc','sc_w','clock_speed','talk_time']]
feature_scaler = StandardScaler()
X_scaled = feature_scaler.fit_transform(X_)


#####################################################################################
# Implementing Random Forest Regression
# Tuning the random forest parameter 'n_estimators' and implementing cross-validation using Grid Search
rfr = RandomForestRegressor(criterion='squared_error', max_features='sqrt', random_state=1)
grid_param = {'n_estimators': [50,100,150,200,250,300]}

gd_sr = GridSearchCV(estimator=rfr, param_grid=grid_param, scoring='r2', cv=5)

gd_sr.fit(X_scaled, Y)

best_parameters = gd_sr.best_params_
print("Best parameters: ", best_parameters)

best_result = gd_sr.best_score_ # Mean cross-validated score of the best_estimator
print("r2: ", best_result)

Adj_r2 = 1-(1-best_result)*(1600-1)/(1600-16-1)
print("Adjusted r2: ", Adj_r2)

'''
Adj_r2 = 1-(1-r2)*(n-1)/(n-p-1)

where, n = number of observations in training data, p = number of features
```

- We get the below result:

```
Best parameters:  {'n_estimators': 300}
r2:  0.7955330021957584
Adjusted r2:  0.7934663742962841
```

- We can see that we get an even better score in this compared to all other models, hence we can say that there was some nonlinearity in our data set but we random forest regressor we overcame that and we got the best result.

Conclusion:

For the above interpretation, we can see that all the models performed equally well and gave the best result but the score given by random forest regressor was better than all of those, hence we would go with random forest regressor.