DEPARTMENT OF MATHEMATICS AND STATISTICS, IIT KANPUR

Artificial Intelligence Project Work

Clustering and Classification of Mobile Price Range on "Mobile Price" dataset

Submitted to:-

Dr. Amit Mitra

Submitted By:-

Ankit Gupta

191016

Acknowledgement

I would like to express my special thanks of gratitude to my teacher ***Dr. Amit Mitra*** *who gave me the golden opportunity to do this wonderful project on the topic*

*"**Clustering and Classification of Mobile Price Range on "Mobile Price" dataset"***

which also helped me in doing a lot of Research and i came to know about so many new things I am really thankful to them.
Secondly i would also like to thank my seniors, friends and others who helped me a lot in finalizing this project within the limited time frame

CONTENTS

AIM: -

We have **mobile price classification** data set. I want to divide data into different cluster using K mean clustering , check the performance of different statistical classification technique on given data and then use best method to predict the class level of test data set.

INTRODUCTION: -

Nowadays Mobile is most common and essential commodity. Everyone wants to know the price range of mobile phone whenever he/she want to buy a mobile phone.
In this project work I mainly focused on predicting the mobile price classes using suitable statistical techniques of classification with the help of given "Training Data Set".
In this problem you do not have to predict actual price but a price range indicating how high the price is.

I will deal this problem using R SOFTWARE.

ABOUT DATA: -

I have taken data**” Mobile Price Classification”** from **KAGGLE**.

In this data we have two data set, one is **Training Data** and other is **Test Data.** In Training set I have response variable but in test set I have not response variable. I have to make prediction of price range of test set by making study on training set. For doing so I will split training data into two part and one part will considered as ”Training” and second as ”Test”. And which statistical technique will perform best on this test set will use for classification for real test set for which we have to predict class level.

DATA SOURCE: -

https://www.kaggle.com/iabhishekofficial/mobile-price-classification

TRAINING DATA: -

https://www.kaggle.com/iabhishekofficial/mobile-price- classification ?select=train.csv

TEST DATA: -

https://www.kaggle.com/iabhishekofficial/mobile-price- classification ?select=test.csv

Data is of the form:-

battery_pc	blue	clock_spe	dual_sim	fc	four_g	int_memo	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_scr	wifi	price_range
842	0	2.2	0	1	0	7	0.6	188	2	2	20	756	2549	9	7	19	0	0	1	1
1021	1	0.5	1	0	1	53	0.7	136	3	6	905	1988	2631	17	3	7	1	1	0	2
563	1	0.5	1	2	1	41	0.9	145	5	6	1263	1716	2603	11	2	9	1	1	0	2
615	1	2.5	0	0	0	10	0.8	131	6	9	1216	1786	2769	16	8	11	1	0	0	2
1821	1	1.2	0	13	1	44	0.6	141	2	14	1208	1212	1411	8	2	15	1	1	0	1
1859	0	0.5	1	3	0	22	0.7	164	1	7	1004	1654	1067	17	1	10	1	0	0	1
1821	0	1.7	0	4	1	10	0.8	139	8	10	381	1018	3220	13	8	18	1	0	1	3
1954	0	0.5	1	0	0	24	0.8	187	4	0	512	1149	700	16	3	5	1	1	1	0
1445	1	0.5	0	0	0	53	0.7	174	7	14	386	836	1099	17	1	20	1	0	0	0
509	1	0.6	1	2	1	9	0.1	93	5	15	1137	1224	513	19	10	12	1	0	0	0
769	1	2.9	1	0	0	9	0.1	182	5	1	248	874	3946	5	2	7	0	0	0	3
1520	1	2.2	0	5	1	33	0.5	177	8	18	151	1005	3826	14	9	13	1	1	1	3
1815	0	2.8	0	2	0	33	0.6	159	4	17	607	748	1482	18	0	2	1	0	0	1
803	1	2.1	0	7	0	17	1	198	4	11	344	1440	2680	7	1	4	1	0	1	2
1866	0	0.5	0	13	1	52	0.7	185	1	17	356	563	373	14	9	3	1	0	1	0
775	0	1	0	3	0	46	0.7	159	2	16	862	1864	568	17	15	11	1	1	1	0

In our data set there are 20 independent variable say $x_1,x_2,x_3,. \ldots\ldots\ldots\ldots$ 20 by name battery power, internal memory, mobile weight, ram etc. Complete list of independent variable is given as..

```
> # to seeing the data set
> colnames(train_data)
 [1] "battery_power" "blue"          "clock_speed"   "dual_sim"      "fc"            "four_g"        "int_memory"
 [8] "m_dep"         "mobile_wt"     "n_cores"       "pc"            "px_height"     "px_width"      "ram"
[15] "sc_h"          "sc_w"          "talk_time"     "three_g"       "touch_screen"  "wifi"          "price_range"
> |
```

So, in our data set, response variable(y), is price range whose level are ’0’,’1’,’2’,’3’ and these are” low cost”,” medium cost”,” high cost”, very high cost” respectively.

class level	class level name
0	low cost
1	medium cost
2	high cost
3	very high cost

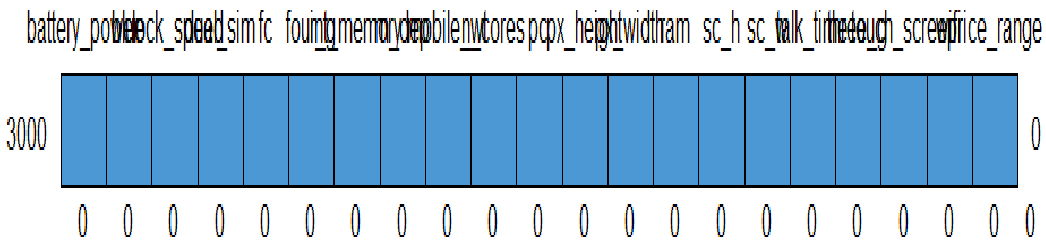
Originally, we have 2000 observation into training data set and 1000 observation in test data set.

Manipulation of Data: -

First, we will check for missing value in the data and if it occurs than will remove corresponding rows. As we have quite large data set so removing corresponding row of missing row is not bad option for dealing with missing observations.(In case of small data set we can predict missing observation by different technique).

```
> # searching for missing value
> md.pattern(combi)     # no missing value find
 /\     /\
{  `---'  }
{  O   O  }
==>  V <==  No need for mice. This data set is completely observed.
 \  \|/  /
  `-----'

     battery_power blue clock_speed dual_sim fc four_g int_memory m_dep mobile_wt n_cores pc px_height px_width ram sc_h sc_w
3000             1    1           1        1  1      1          1     1         1       1  1         1        1   1    1    1
                 0    0           0        0  0      0          0     0         0       0  0         0        0   0    0    0
     talk_time three_g touch_screen wifi price_range
3000         1       1            1    1           1 0
             0       0            0    0           0 0
```

Fortunately, there is no missing value in our data set.



Now we will look for categorical variable and will convert that in numeric variable, and clearly from image, all variable is in numeric form in our data set.

```
> str(my_data) # There is no need of changing catogorical column into numeric as all predictors are numeric
'data.frame':	3000 obs. of  20 variables:
 $ battery_power: int  842 1021 563 615 1821 1859 1821 1954 1445 509 ...
 $ blue         : int  0 1 1 1 1 0 0 0 1 1 ...
 $ clock_speed  : num  2.2 0.5 0.5 2.5 1.2 0.5 1.7 0.5 0.5 0.6 ...
 $ dual_sim     : int  0 1 1 0 0 1 0 1 0 1 ...
 $ fc           : int  1 0 2 0 13 3 4 0 0 2 ...
 $ four_g       : int  0 1 1 0 1 0 1 0 0 1 ...
 $ int_memory   : int  7 53 41 10 44 22 10 24 53 9 ...
 $ m_dep        : num  0.6 0.7 0.9 0.8 0.6 0.7 0.8 0.8 0.7 0.1 ...
 $ mobile_wt    : int  188 136 145 131 141 164 139 187 174 93 ...
 $ n_cores      : int  2 3 5 6 2 1 8 4 7 5 ...
 $ pc           : int  2 6 6 9 14 7 10 0 14 15 ...
 $ px_height    : int  20 905 1263 1216 1208 1004 381 512 386 1137 ...
 $ px_width     : int  756 1988 1716 1786 1212 1654 1018 1149 836 1224 ...
 $ ram          : int  2549 2631 2603 2769 1411 1067 3220 700 1099 513 ...
 $ sc_h         : int  9 17 11 16 8 17 13 16 17 19 ...
 $ sc_w         : int  7 3 2 8 2 1 8 3 1 10 ...
 $ talk_time    : int  19 7 9 11 15 10 18 5 20 12 ...
 $ three_g      : int  0 1 1 1 1 1 1 1 1 1 ...
 $ touch_screen : int  0 1 1 0 1 0 0 1 0 0 ...
 $ wifi         : int  1 0 0 0 0 0 1 1 0 0 ...
```

as correlation among independent variable is very low (for al- most all pairs), so all predictors (x_1,x_2,x_3,......x_{20}) are almost independent to each other so at this stage we can not remove any independent variable.

From the image below it can be seen that correlation among variable is very low

```
> cor(train)   # as correlation between predictors is very less so we can not remove any variable in this stage
              battery_power         blue  clock_speed     dual_sim           fc       four_g   int_memory        m_dep
battery_power   1.000000000  0.0333888679  0.006889263 -0.039003260  0.014897061  0.033237333  0.010694634  0.022355088
blue            0.033388868  1.0000000000  0.004900940  0.044635178 -0.010391116  0.010141889  0.031744457  0.006254646
clock_speed     0.006889263  0.0049009403  1.000000000  0.003621465 -0.030795779 -0.029098476  0.001100561 -0.009584948
dual_sim       -0.039003260  0.0446351779  0.003621465  1.000000000 -0.020875460  0.004492827 -0.025816126 -0.020694033
fc              0.014897061 -0.0103911165 -0.030795779 -0.020875460  1.000000000 -0.023500056 -0.030624648  0.011275328
four_g          0.033237333  0.0101418889 -0.029098476  0.004492827 -0.023500056  1.000000000  0.001684709 -0.011167576
int_memory      0.010694634  0.0317444572  0.001100561 -0.025816126 -0.030624648  0.001684709  1.000000000  0.008224110
m_dep           0.022355088  0.0062546459 -0.009584948 -0.020694033  0.011275328 -0.011167576  0.008224110  1.000000000
mobile_wt       0.011517674 -0.0041096553  0.025603581  0.008186739  0.031846857 -0.029874841 -0.026125036  0.015113507
n_cores        -0.043770782  0.0328370794  0.017220150 -0.031970105 -0.015368806 -0.039616734 -0.027146159  0.007560423
pc              0.012629997 -0.0149773237 -0.025816675 -0.026924661  0.642982431 -0.024918106 -0.042956527  0.037460557
px_height       0.021162222  0.0005325944 -0.016511542 -0.023123388  0.003681564 -0.025249824 -0.004632229  0.033567911
px_width       -0.011397183 -0.0336015184 -0.023579405  0.011900663  0.019197994 -0.010025719 -0.014136047  0.035626718
ram             0.007686835  0.0618821951  0.002990794  0.034701449  0.017509899 -0.014032094  0.026430853 -0.029312522
sc_h           -0.040506439 -0.0046408898 -0.047122211 -0.014303918 -0.017929412  0.017062905  0.022426608 -0.042162340
sc_w           -0.018402627  0.0095972372 -0.007943719 -0.024038979 -0.010838074  0.047470727 -0.007777258 -0.012421559
talk_time       0.059009056  0.0091759763 -0.004051084 -0.050175357  0.026194098 -0.048638973 -0.017169523  0.019943324
three_g         0.032276788 -0.0400372876 -0.037848474 -0.012458821 -0.017917881  0.577820360 -0.026825025 -0.043601563
touch_screen   -0.009884851  0.0113788393  0.022767113 -0.028764717 -0.040438111  0.021710291 -0.011807883 -0.014305094
wifi           -0.007253740 -0.0179755773 -0.022044849  0.011265350  0.030692765 -0.020998478 -0.015512790 -0.038000307
price_range     0.213901926  0.0618433846 -0.007686164  0.004452335  0.023132963 -0.003822645  0.044683163 -0.024022857
                 mobile_wt      n_cores           pc     px_height     px_width          ram         sc_h         sc_w
battery_power  0.0115176743 -0.043770782  0.012629997  0.0211622216 -0.011397183  0.007686835 -0.040506439 -0.018402627
blue          -0.0041096553  0.032837079 -0.014977324  0.0005325944 -0.033601518  0.061882195 -0.004640890  0.009597237
clock_speed    0.0256035811  0.017220150 -0.025816675 -0.0165115416 -0.023579405  0.002990794 -0.047122211 -0.007943719
dual_sim       0.0081867390 -0.031970105 -0.026924661 -0.0231233882  0.011900663  0.034701449 -0.014303918 -0.024038979
fc             0.0318468573 -0.015368806  0.642982431  0.0036815635  0.019197994  0.017509899 -0.017929412 -0.010838074
four_g        -0.0298748414 -0.039616734 -0.024918106 -0.0252498242 -0.010025719 -0.014032094  0.017062905  0.047470727
int_memory    -0.0261250355 -0.027146159 -0.042956527 -0.0046322295 -0.014136047  0.026430853  0.022426608 -0.007777258
m_dep          0.0151135068  0.007560423  0.037460557  0.0335679111  0.035626718 -0.029312522 -0.042162340 -0.012421559
mobile_wt      1.0000000000 -0.034378627  0.042780648 -0.0008195846 -0.004202288 -0.018987844 -0.019797028 -0.025850868
n_cores       -0.0343786274  1.000000000  0.008898856 -0.0199585310  0.024089455  0.017204464  0.002371109  0.015979437
pc             0.0427806482  0.008898856  1.000000000 -0.0061168837  0.023695883  0.031031676  0.018521563 -0.025978846
px_height     -0.0008195846 -0.019958531 -0.006116884  1.0000000000  0.524846648 -0.014815038  0.054045827  0.035415952
px_width      -0.0042022876  0.024089455  0.023695883  0.5248466476  1.000000000  0.007451710  0.020474433  0.045050366
ram           -0.0189878441  0.017204464  0.031031676 -0.0148150380  0.007451710  1.000000000  0.022280273  0.048740318
sc_h          -0.0197970279  0.002371109  0.018521563  0.0540458273  0.020474433  0.022280273  1.000000000  0.497557030
sc_w          -0.0258508676  0.015979437 -0.025978846  0.0354159519  0.045050366  0.048740318  0.497557030  1.000000000
talk_time      0.0065299881  0.021854989  0.043077436  0.0030441017  0.015883984  0.009110133 -0.022373634 -0.031969464
three_g       -0.0147426079 -0.015584347 -0.010727473 -0.0442589784 -0.030987634 -0.011552381  0.015945910  0.038856314
touch_screen  -0.0199115546  0.030778438 -0.039388850  0.0186665274  0.002847034 -0.032223530 -0.012679918  0.024377091
wifi          -0.0010523774 -0.031121026  0.010728123  0.0594554572  0.029418782  0.018855082  0.018310609  0.034880197
price_range   -0.0481404984  0.012084140  0.036869024  0.1535785245  0.173007136  0.914593696  0.020799530  0.050470109

                 talk_time      three_g touch_screen         wifi  price_range
battery_power  0.059009056  0.032276788 -0.009884851 -0.007253740  0.213901926
blue           0.009175976 -0.040037288  0.011378839 -0.017975577  0.061843385
clock_speed   -0.004051084 -0.037848474  0.022767113 -0.022044849 -0.007686164
dual_sim      -0.050175357 -0.012458821 -0.028764717  0.011265350  0.004452335
fc             0.026194098 -0.017917881 -0.040438111  0.030692765  0.023132963
four_g        -0.048638973  0.577820360  0.021710291 -0.020998478 -0.003822645
int_memory    -0.017169523 -0.026825025 -0.011807883 -0.015512790  0.044683163
m_dep          0.019943324 -0.043601563 -0.014305094 -0.038000307 -0.024022857
mobile_wt      0.006529988 -0.014742608 -0.019911555 -0.001052377 -0.048140498
n_cores        0.021854989 -0.015584347  0.030778438 -0.031121026  0.012084140
pc             0.043077436 -0.010727473 -0.039388850  0.010728123  0.036869024
px_height      0.003044102 -0.044258978  0.018666527  0.059455457  0.153578524
px_width       0.015883984 -0.030987634  0.002847034  0.029418782  0.173007136
ram            0.009110133 -0.011552381 -0.032223530  0.018855082  0.914593696
sc_h          -0.022373634  0.015945910 -0.012679918  0.018310609  0.020799530
sc_w          -0.031969464  0.038856314  0.024377091  0.034880197  0.050470109
talk_time      1.000000000 -0.041077031  0.017880602 -0.024067977  0.025050371
three_g       -0.041077031  1.000000000  0.027363543 -0.007091468 -0.003902526
touch_screen   0.017880602  0.027363543  1.000000000  0.015152594 -0.029545336
wifi          -0.024067977 -0.007091468  0.015152594  1.000000000  0.015842874
price_range    0.025050371 -0.003902526 -0.029545336  0.015842874  1.000000000
```

Principal Component Analysis: -

PCA is unsupervised method of dealing with data or visualization of data. It is also a tool used for data pre-processing be- fore supervised techniques are applied. When we have a larger no correlated variables, Principal Component allow us to summaries the data in lower dimension along principal component direction. And principal component directions are direction along which variables are highly variable.

PCA is a method by which we compute principal component, and it is a method which used to reduced dimension of data set and represent data in lower dimension with maximum variability. First PC define maximum variability along it's direction, second PC define second most variability along it's direction and so on. Total no of principal component for a data is min(n-1,p) where n is no of observation and p dimension of data vector. while computing principal component we used to normalized feature space and Ith principal component is linear combination of normalized feature space which is given as

$Z_i=\varphi_{1i}X_1+\varphi_{2i}X_2+\varphi_{3i}X_3+ \ldots\ldots\ldots\ldots\ldots \varphi_{pi}X_p$

Where Z_i is ith principal component and $\varphi_{1i}, \varphi_{2i},\ldots, \varphi_{pi}$ are loadings of ith principal component.

Under" ggfortify" package "pccomp()" provide principal compo- nent on normalized data by default. From below this is clear that it has five components, where" sdev" is standard deviation of PC ,"rotation" is matrix of loadings and "x" is matrix of principal component score vector.

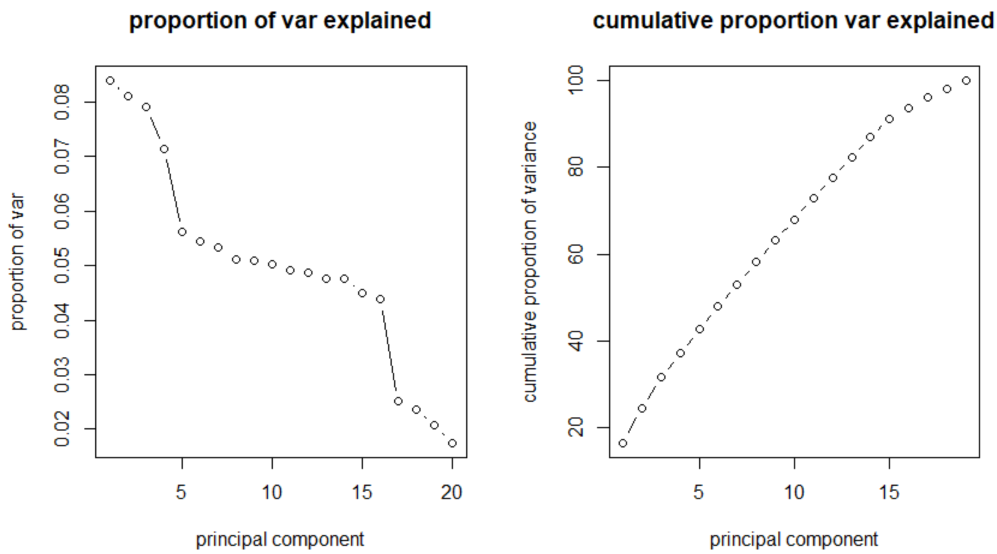
```
> library(ggfortify)
> #The base R function prcomp() is used to perform PCA. By default, it centers the variable to have mean equals to zero.
> #With parameter scale. = T, we normalize the variables to have standard deviation equals to 1.
> pc=prcomp(train,scale. = T)
> names(pc)
[1] "sdev"     "rotation" "center"   "scale"    "x"
> summary(pc)
Importance of components:
                           PC1     PC2     PC3     PC4     PC5     PC6     PC7     PC8     PC9    PC10    PC11    PC12    PC13
Standard deviation     1.29509 1.27377 1.25800 1.19543 1.05901 1.04266 1.03230 1.01016 1.00848 1.00284 0.99031 0.98734 0.97583
Proportion of Variance 0.08386 0.08112 0.07913 0.07145 0.05608 0.05436 0.05328 0.05102 0.05085 0.05028 0.04904 0.04874 0.04761
Cumulative Proportion  0.08386 0.16499 0.24412 0.31557 0.37164 0.42600 0.47928 0.53030 0.58116 0.63144 0.68048 0.72922 0.77683
                          PC14    PC15    PC16    PC17    PC18    PC19    PC20
Standard deviation     0.97553 0.94807 0.93601 0.70768 0.68724 0.64252 0.59233
Proportion of Variance 0.04758 0.04494 0.04381 0.02504 0.02361 0.02064 0.01754
Cumulative Proportion  0.82441 0.86935 0.91316 0.93820 0.96182 0.98246 1.00000
```

It is also clear that 1st PC explain around 8.38 percent and 2nd 8.11 percent variability of total variability, which are very less variability in comparison of total variability.

So, PCA is not good technique for data handling in this case which can also be seen from less correlation among variables.

From below it is clear that first 13 PC explain around 82 per- cent variability which is not considerable dimension reduction

```
> pc_var=(pc$sdev)^2
> mat=matrix(rep(0,2*ncol(my_data)),ncol = 2)
> mat[1,1]="no of pc"
> mat[1,2]="cumulative proportion var explained"
> for (i in 2:ncol(my_data)){
+   mat[i,1]=i-1
+   mat[i,2]=100*sum(pc_var[1:i])/sum(pc_var)
+ }
> mat
      [,1]       [,2]
 [1,] "no of pc" "cumulative proportion var explained"
 [2,] "1"        "16.4987628942819"
 [3,] "2"        "24.411617145793"
 [4,] "3"        "31.556842094238"
 [5,] "4"        "37.1643975246934"
 [6,] "5"        "42.6000794435989"
 [7,] "6"        "47.9283005317594"
 [8,] "7"        "53.0303920355636"
 [9,] "8"        "58.1156011749805"
[10,] "9"        "63.144027931119"
[11,] "10"       "68.0476469164519"
[12,] "11"       "72.9218070683758"
[13,] "12"       "77.6829992407642"
[14,] "13"       "82.4412454013051"
[15,] "14"       "86.9354133244602"
[16,] "15"       "91.3160086486058"
[17,] "16"       "93.8200303983614"
[18,] "17"       "96.1815288404238"
[19,] "18"       "98.2457203939736"
[20,] "19"       "100"
```



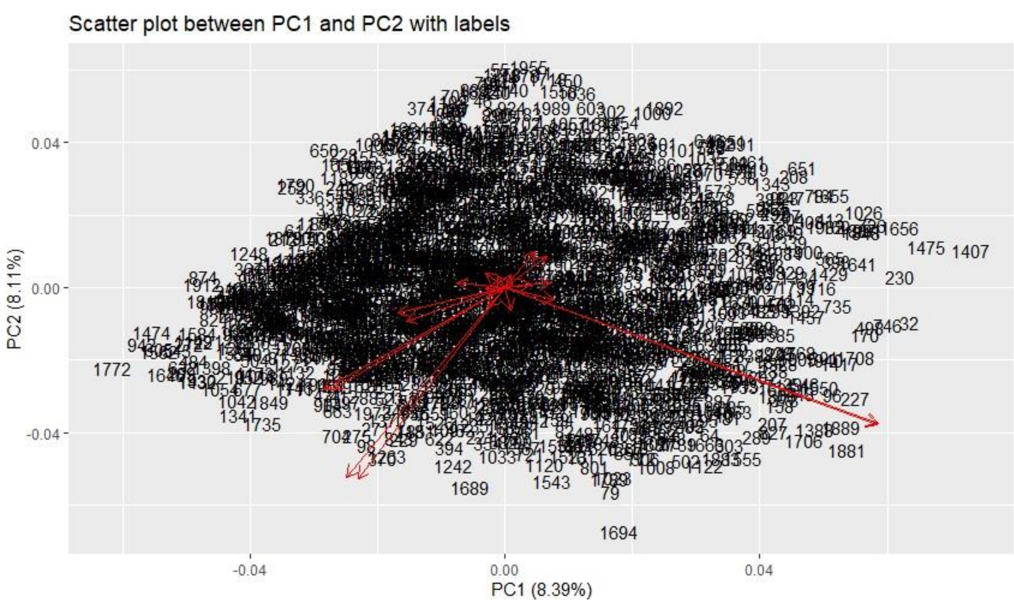
From **"Cumulative proportion variance explained"** it is clear that rate of increasing "cumulative proportion of variance" is approximately same for most of the PC so it implies that not even first PC explaining large variance.

Clustering Using K Mean: -

Clustering we mean divide the set of observation into subgroups such that observation within subgroup are similar as possible as.So clustering looks for homogeneous subgroup.

In k mean clustering we divide the observation into k groups where k, the no of subgroups, we know in advance. Here clusters are such that each observation belongs to at least one of thecluster and non-overlapping. Here within cluster variation should be small as possible as.

From figure it is clear that three points are outliers, so we first remove that point. Also as data is very aggregated, no clear indication for number of cluster k, so normally we apply K mean clustering with k=3.



In R ”kmeans()” command used for performing K-mean c clustering. n this command, we use an argument ”nstart=” and if we use ”nstart=” greater than one then it uses multiple random assignment for clustering and ”kmeans()” perform best result. Generally we use “nstart” greater than 20 which will provide better results.

When we use “**nstart=40**” results are below:-

```
> set.seed(2)
> kmean1=kmeans(train,3,nstart=40)     # no of cluster is three
> total=kmean1$totss
> kmean1$withinss
[1] 467228006 504328670 436997197
> within=kmean1$tot.withinss
> between=kmean1$betweenss
> kmean1$size
[1] 683 705 609
> within_prop=(within/total)*100
> bet_prop=(between/total)*100
> within_prop
[1] 40.21473
> bet_prop
[1] 59.78527
```

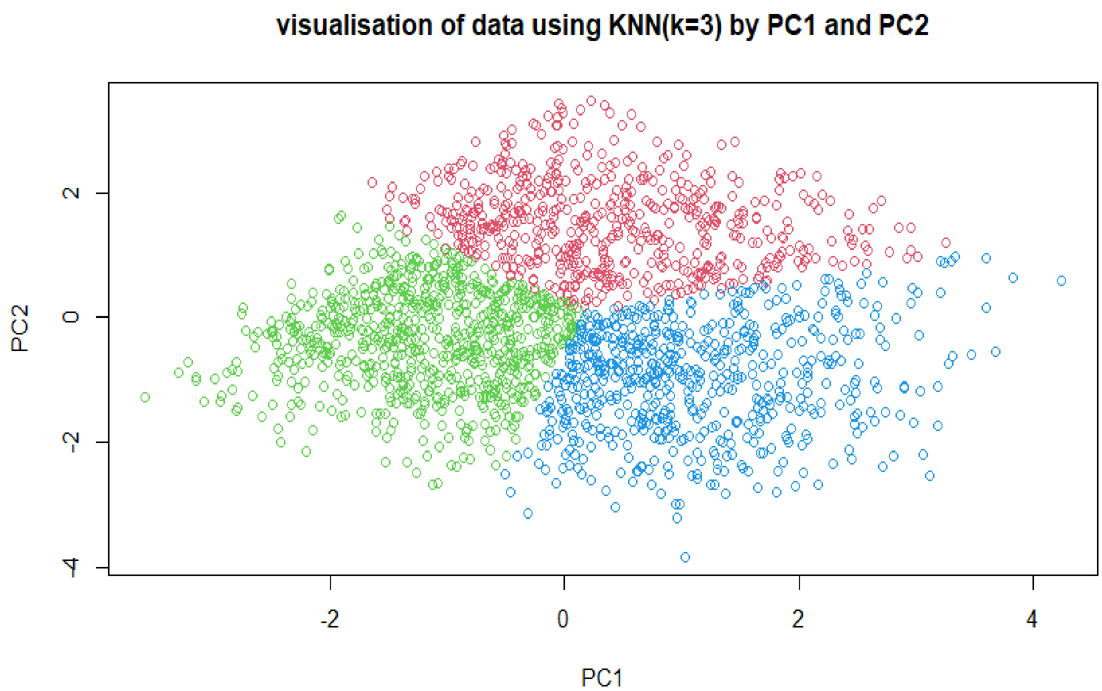
here ”totss”, ”tot.withness”, ”betweenss” are components of ”kmeans()” which gives total sum of square, total within cluster sum of square, and between cluster sum of square respectively. So in this case, within cluster sum of square distance is around 40.41 percent of total sum of square distance. And this puts 683,705,609 observation in three subgroups.

Now we will use 1st two PCA for k mean clustering

```
> pc1=prcomp(train,scale. = T)
> Y=pc$x[,1:2]
> kmean2=kmeans(Y,3,nstart = 40)
> par(mfrow=c(1,1))
> plot(Y,col=(kmean2$cluster+1),main="visualisation of data using KNN(k=3) by PC1 and PC2")
> total2=kmean2$totss
> kmean2$withinss
[1] 819.3192 860.4233 927.2505
> within2=kmean2$tot.withinss
> between2=kmean2$betweenss
> kmean2$size
[1] 589 787 624
> within_prop2=(within2/total2)*100
> bet_prop2=(between2/total2)*100
> within_prop2
[1] 39.52262
> bet_prop2
[1] 60.47738
```

from this picture it is clear that now the observations divided into three parts having size respectively 589,787,624.And the very important thing is that the proportion of total distance within cluster is around 39 percent which is smaller than the earlier case so this method improves clusters.

Visualization of clusters using first two PC



CLASSIFICATION:-

Classification is a process in which we predict a categorical/qualitative response, and predicting the qualitative response for an observation is known as classifying the observation, and response of qualitative variable is known as class levels. In our case **Price Range** is qualitative response, having four classes/categories. Here Class level are ’0’,’1’,’2’,’3’ corresponding to” low cost”,” medium cost”,” high cost”, ” very high cost” respectively.

For dealing this problem we divide the train data into 3:1(1500:500) and bigger portion treated as training set on which we fit our different model and smaller portion of this training set is consider c as test set. we will apply different statistical techniques for classification, and choose one of that for which **classification error or misclassification.** is minimum.

While dealing this problem, I use **k fold cross validation where k=500**,in which, within original training set I have considered 500 different training and test set. And I calculate misclassification error for all test set and then average out these errors. For comparing results, I also apply opposite technique where train size is 500 and test size is 1500.

CLASSIFICATION using PCA: -

Although PCA is unsupervised technique, still it can be used for supervised purpose like quantitative prediction, classification. In supervised technique PCA used as a tool for datapre-processing. For doing so in this case we consider first 13 PC as these explain more than 80 percent variability.

```
> # partion of training set into 3:1
> n=nrow(train.data)
> n1=n*3/4
> n2=n*1/4
> error1=rep(0,n2)
> # when the training set has size 1500 and test set has 500
> for(i in 1:n2){
+   train1=train.data[i:(n1+i-1),]
+   test1=train.data[-(i:(n1+i-1)),]
+   model=rpart(price_range ~ .,data = train.data, method = "anova")
+   Y_cap=predict(model,newdata=test1)
+   Y_cap=round(Y_cap)
+   Y=subset(test1,select=c(price_range))
+   Y=Y[,1]
+
+   k1=0
+   for(j1 in 1:n2){
+     if(Y_cap[j1]!=Y[j1]){
+       k1=k1+1
+     }
+   }
+   error1[i]=(1/n2)*k1
+ }
> test_error_PCA1=mean(error1)
> # when the training set has size 500 and test set has 1500
> error2=rep(0,n2)
> for(i in 1:n2){
+   train1=train.data[i:(n2+i-1),]
+   test1=train.data[-(i:(n2+i-1)),]
+   model=rpart(price_range ~ .,data = train.data, method = "anova")
+   Y_cap=predict(model,newdata=test1)
+   Y_cap=round(Y_cap)
+   Y=subset(test1,select=c(price_range))
+   Y=Y[,1]
+
+   k2=0
+   for(j2 in 1:n1){
+     if(Y_cap[j2]!=Y[j2]){
+       k2=k2+1
+     }
+   }
+   error2[i]=(1/n1)*k2
```

From below it is clear than misclassification test error is quite small when training set have size large(size =1500) as we except. **Still Misclassification rate is around 50 percent which is not quite good and we can expect this as we already discuss that PCA is not good technique for this data set.**

```
> mat1
     [,1]
[1,] "missclassification test error when training set size is 1500"
[2,] "0.503748"
     [,2]
[1,] "missclassification test error when training set size is 500"
[2,] "0.507732"
> test_error_PCA=test_error_PCA1
> test_error_PCA
[1] 0.503748
```

CLASSIFICATION using MULTICLASS LOGESTIC DISCRIMINATION by NEURAL NETWORK

As we have four class level for classification so I have used multiple logistic discrimination. First, I remove some predictors which are nor significant using p value.

```
> # now we will comput valaue using z score
> z= k2$coefficients/k2$standard.errors
> z
  (Intercept) battery_power       blue clock_speed  dual_sim         fc     four_g int_memory      m_dep  mobile_wt
1   -144214.3      46.65122 -0.7181902   -4.523174 -7.505016 -0.4945771  -6.231149   1.895512   9.421581  -7.664001
2   -149581.2      76.92284 -0.6536854   -3.370501 -8.723648 -0.9052611 -14.048409   4.148242 -15.409283 -10.980882
3   -375623.5     101.09836 -24.8920281  -5.601232 -4.097727 -1.3498957 -14.125337   7.004988 -21.371771 -16.135161
   n_cores         pc px_height px_width       ram      sc_h        sc_w  talk_time    three_g touch_screen      wifi
1 3.123417 0.09551272  23.63633 20.31794  63.28658 -1.045643 -0.73969516 -2.5560385 -4.4476728   -0.8245411  -8.98302
2 4.494546 1.98908277  33.37218 33.97499 106.03292 -1.295277  0.06458352 -1.5500778  0.9200333   -4.4318919 -18.45826
3 5.020674 2.47245789  48.85665 44.46684 149.46268  0.965293  0.90447271 -0.9829227 -1.4482460  -13.8752138 -44.90136
> p_val=2*(pnorm(abs(z),lower.tail = F))
> p_val   # clearly there are manr pridictors that are significant
  (Intercept) battery_power         blue  clock_speed     dual_sim        fc       four_g   int_memory        m_dep
1           0             0  4.726400e-01 6.091919e-06 6.142131e-14 0.6208986 4.630250e-10 5.802463e-02  4.443476e-21
2           0             0  5.133145e-01 7.503152e-04 2.693786e-18 0.3653271 7.878433e-45 3.350382e-05  1.417751e-53
3           0             0 9.076420e-137 2.128335e-08 4.172277e-05 0.1770494 2.651287e-45 2.470077e-12 2.446759e-101
     mobile_wt      n_cores         pc     px_height      px_width ram      sc_h      sc_w  talk_time      three_g
1 1.802281e-14 1.787641e-03 0.92390758 1.631456e-123  8.924655e-92   0 0.2957260 0.4594850 0.01058714 8.680562e-06
2 4.722891e-28 6.971855e-06 0.04669207 3.473605e-244 5.216856e-253   0 0.1952248 0.9485056 0.12112284 3.575554e-01
3 1.444312e-58 5.149044e-07 0.01341875  0.000000e+00  0.000000e+00   0 0.3343981 0.3657448 0.32564551 1.475483e-01
  touch_screen         wifi
1 4.096322e-01 2.634355e-19
2 9.340985e-06 4.475224e-76
3 8.952570e-44 0.000000e+00
> print("from p value clearly of blue,fc,pc, sc_h,sc_w,talk_time are not significant")
[1] "from p value clearly of blue,fc,pc, sc_h,sc_w,talk_time are not significant"
```

We can apply Multiple Logistic Discrimination by ”multinom()” command under ”nnet” library After removing some insignificant variable we have 15 variable and now we can apply Multiple Logistic Discrimination using **k fold cross validation where k=500**

```
> combine_data=subset(combine_data,select=-c(blue,fc,pc, sc_h,sc_w,talk_time))
> ncol(combine_data)
[1] 15
> train=combine_data[1:nrow(data_train),]
> test=combine_data[-(1:nrow(data_train)),]
> n1=nrow(train)*(3/4)    # n1 is size of trainig set
> n2=nrow(train)*(1/4)    # n2 is size of test set
> # size of trainig set is 1500 1nd test set is 500
> error1=rep(0,n2)
> for(i in 1:n2){
+   train1=train[i:(n1+i-1),]
+   test1=train[-(i:(n1+i-1)),]
+   model=multinom(price_range~.,data=train1)
+   Y_cap=predict(model,newdata=test1)
+   Y=subset(test1,select=c(price_range))
+   Y=Y[,1]
+   k=0
+   for(j in 1:n2){
+     if(Y_cap[j]!=Y[j]){
+       k=k+1
+     }
+   }
+   error1[i]=(1/n2)*k
+ }
```

Now we have to compute misclassification rate'

```
> test_error_log2=mean(error2)
> mat1=matrix(rep(0,4),ncol = 2,byrow=T)
> mat1[1,1]="missclassification error when training set size is 1500"
> mat1[1,2]="missclassification error when training set size is 500"
> mat1[2,1]=test_error_log1
> mat1[2,2]=test_error_log2
> mat1
     [,1]                                                       [,2]
[1,] "missclassification error when training set size is 1500" "missclassification error when training se
[2,] "0.025164"                                                 "0.053544"
> test_error_log=test_error_log1
> test_error_log
[1] 0.025164
```

From here it can be seen that **Misclassification rate is very low for that is around 2.51 percent**

Also, from matrix(mat1) it can be seen that misclassification rate is smaller when training set size is large (size=1500) as we expected. **As test error is very small in this scenario so this model fit data very well and can provide very good prediction for original Test Set**

CLASSIFICATION using SVM

Support Vector Machine is another powerful technique for misclassification may provide very good result in some cases.
we can apply SVM technique to our data using" svm()" command with suitable arguments under "e1071" library. I have applied SVM to our data under **"polynomil" and "radial" kernel** for different degree in polynomial kernel and in radial kernel. **In this case I take a random training set of size 1500.** for fitting SVM model

When we applied "polynomial" kernel to svm

```
> #when kernal is polynomial
> svmfit_poly=svm(y~.,data=dat,kernel="polynomial",ranges=list(cost=c(0.1,1,10)),degree=c(4,6,8))
> tune_poly=tune(svm,y~.,data=dat,kernel="polynomial",ranges=list(cost=c(0.1,1,10)),degree=c(4,6,8))
> bestmod_poly=tune_poly$best.model
> summary(bestmod_poly)

Call:
best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost = c(0.1, 1, 10)), kernel = "polynomial",
    degree = c(4, 6, 8))


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  polynomial
       cost:  10
     degree:  4 6 8
     coef.0:  0

Number of Support Vectors:  1458

 ( 368 368 347 375 )


Number of Classes:  4

Levels:
 0 1 2 3



> ypred_poly=predict(bestmod_poly,test_data)
> tab1=table(predict=ypred_poly,true=test_y)
> tab1
       true
predict  0  1  2  3
      0 54 14 13 34
      1 18 64 43 12
      2 23 45 54 30
      3 24  4 11 57
> test_error_svm_poly=1-(sum(diag(tab1)))/sum(tab1)
> test_error_svm_poly
[1] 0.542
```

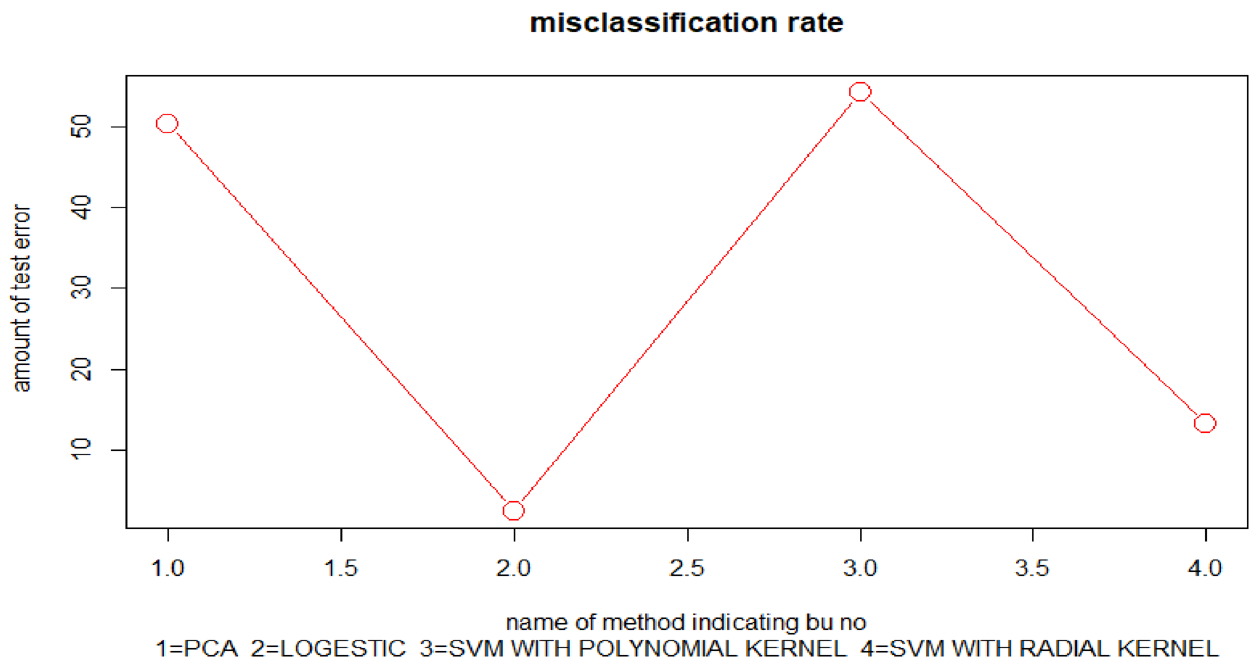
from this picture we can conclude that in "polynomial" kernel cost=10 provide best result. In this set up there are 1458 sup- port vector and **test misclassification rate is around 54.2 percent** which indicate that model is not good under polynomial kernel as test misclassification rate is quite high.

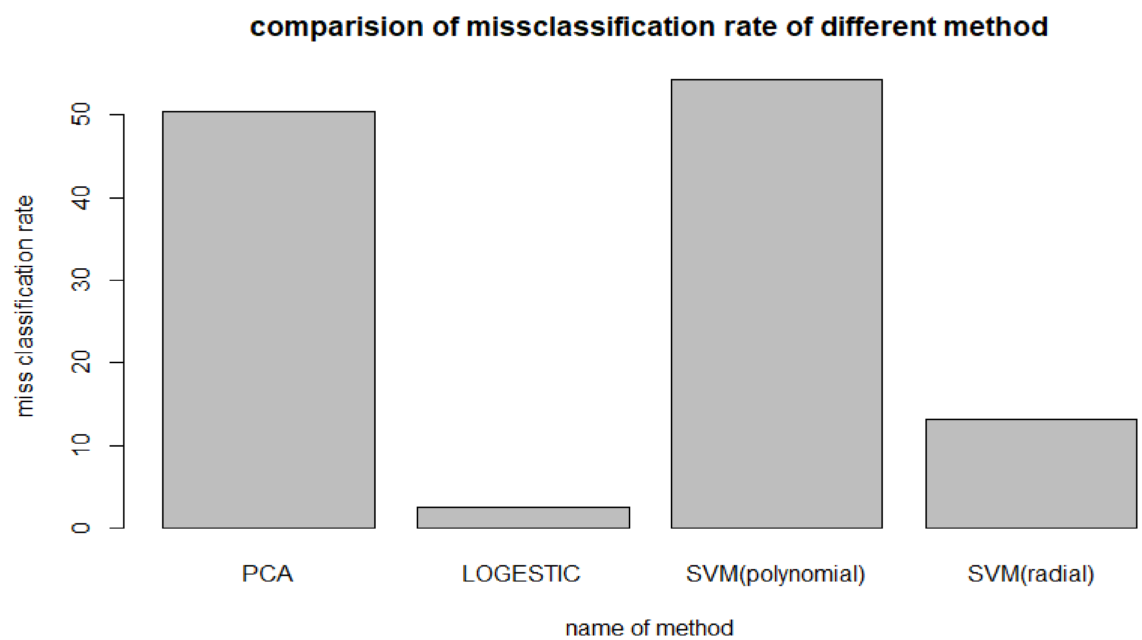
When we applied ”radial” kernel to SVM

```
> #when kernal is radial
> svmfit_rad=svm(y~.,data=dat,kernel="radial",ranges=list(cost=c(0.1,1,10)),degree=c(4,6,8))
> tune_rad=tune(svm,y~.,data=dat,kernel="radial",ranges=list(cost=c(0.1,1,10)),degree=c(4,6,8))
> bestmod_rad=tune_rad$best.model
> summary(bestmod_rad)

Call:
best.tune(method = svm, train.x = y ~ ., data = dat, ranges = list(cost = c(0.1, 1, 10)), kernel = "radial",
    degree = c(4, 6, 8))


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  radial
       cost:  10

Number of Support Vectors:  1088

 ( 202 345 195 346 )


Number of Classes:  4

Levels:
 0 1 2 3



> ypred_rad=predict(bestmod_rad,test_data)
> tab2=table(predict=ypred_rad,true=test_y)
> tab2
       true
predict   0   1   2   3
      0 109   9   0   0
      1  10 106  14   0
      2   0  12  97  11
      3   0   0  10 122
> test_error_svm_rad=1-(sum(diag(tab2)))/sum(tab2)
> test_error_svm_rad
[1] 0.132
```

from this picture we can conclude that in ”radial” kernel cost=10 provide best result. In this set up there are 1088 support vector and **test misclassification rate is around 13.2 percent** which provide better result than ”polynomial” kernel. Even test error is quite low

Comparison of Misclassification Rate





From the misclassification rate graph or bar diagram it is clear that "**MULTICLASS LOGESTIC DISCRIMINATION"**
has minimum misclassification rate. Hence it is best fitting model in our case.
So we will apply MULTICLASS LOGESTIC DISCRIM- INATION method for classification in original Test Set for which we have to predict classes as "0","1,"2","3"

Classification of Test Data by MULTICLASS LOGESTIC DISCRIMINATION

The form of Test data is:-

id	battery_pc	blue	clock_spee	dual_sim	fc	four_g	int_memo	m_dep	mobile_wt	n_cores	pc	px_height	px_width	ram	sc_h	sc_w	talk_time	three_g	touch_s
1	1043	1	1.8	1	14	0	5	0.1	193	3	16	226	1412	3476	12	7	2	0	
2	841	1	0.5	1	4	1	61	0.8	191	5	12	746	857	3895	6	0	7	1	
3	1807	1	2.8	0	1	0	27	0.9	186	3	4	1270	1366	2396	17	10	10	0	
4	1546	0	0.5	1	18	1	25	0.5	96	8	20	295	1752	3893	10	0	7	1	
5	1434	0	1.4	0	11	1	49	0.5	108	6	18	749	810	1773	15	8	7	1	
6	1464	1	2.9	1	5	1	50	0.8	198	8	9	569	939	3506	10	7	3	1	
7	1718	0	2.4	0	1	0	47	1	156	2	3	1283	1374	3873	14	2	10	0	
8	833	0	2.4	1	0	0	62	0.8	111	1	2	1312	1880	1495	7	2	18	0	
9	1111	1	2.9	1	9	1	25	0.6	101	5	19	556	876	3485	11	9	10	1	
10	1520	0	0.5	0	1	0	25	0.5	171	3	20	52	1009	651	6	0	5	1	
11	1500	0	2.2	0	2	0	55	0.6	80	7	6	503	1336	3866	13	7	20	0	
12	1343	0	2.9	0	2	1	34	0.8	171	3	6	235	1671	3911	15	8	8	1	
13	900	1	1.4	1	0	0	30	1	87	2	3	829	1893	439	6	2	20	1	
14	1190	1	2.2	1	5	0	19	0.9	158	5	15	227	1856	992	13	0	16	1	
15	630	0	1.8	0	8	1	51	0.9	193	8	9	1315	1323	2751	17	6	3	1	

```
> #install.packages("nnet")
> library(nnet)
> setwd("C:/Users/hp/OneDrive/Documents/R")
> train_data=read.csv("data mobile price prediction_train.csv")
> test_data=read.csv("data mobile price prediction_test.csv")
> # as test error is minimum for MULTICLASS LOGESTIC DISCRIMINATION so we predict classes for test data using it.
> fit=multinom(price_range~.,data=train_data)
# weights:  88 (63 variable)
initial  value 2772.588722
iter  10 value 2278.873193
iter  20 value 2091.713216
iter  30 value 2022.764974
iter  40 value 1903.676283
iter  50 value 1262.768184
iter  60 value 845.730544
iter  70 value 114.102713
iter  80 value 59.550575
iter  90 value 52.272072
iter 100 value 50.459435
final  value 50.459435
stopped after 100 iterations
> ypredict=predict(fit,newdata = test_data)
> test_summary=summary(ypredict)
> test_summary
  0   1   2   3
249 234 256 261
> test_class_prop=test_summary/1000
> test_class_prop
    0     1     2     3
0.249 0.234 0.256 0.261
> train_resp=train_data[,21]
> train_summary=table(train_resp)
> train_summary
train_resp
  0   1   2   3
500 500 500 500
> train_class_prop=train_summary/2000
> rbind(train_class_prop,test_class_prop)
                     0     1     2     3
train_class_prop 0.250 0.250 0.250 0.250
test_class_prop  0.249 0.234 0.256 0.261
```

From this we can see that, we first fit MULTICLASS LOGES- TIC DISCRIMINATION model to original training set which have 2000 observation and than we predict class levels for test data using this model.
And in training set 25 present observation are in each class”0”,”1”, ”2”,”3”.**Also it can be seen that about 24.9 percent observation in class ”0”,about 23.4 percent observation in class ”1”,about 25.6 percent observation in class ”2”,about 26.1 percent observation in class ”3”.** This result gives us a indication that this model fit the data very well and predict classes with good accuracy.

Conclusion:-

→ There are no missing observation in data and all the factors are Numeric.

→ As First PC not explaining high variance (only 8.38 percent) so other will automatically explain low variance as the result about 13 PC out of 20 are explaining around 82 percent variability. So, PCA is not much helpful in dimension reduction and data visualization in this case.

→ Using K means clustering we divide the data into three sub-groups(k=3). And three subgroups contain 683,705,609 observation and sum of square distance within cluster is around 40.41 percent of total sum of square distance.

→ As we want to minimize sum of square of distance within cluster in K means, so we apply K meansclustering using PCA, and in this case, sum of square distance within cluster is around 39.52 percent of total sum of square distance. So it improves cluster but improvement is not significant. (very less improvement)

→ For classification or for prediction of class level we apply three techniques

(1) Classification Using PCA

(2) Classification using MULTICLASS LOGISTIC DISCRIMINATION

(3) Classification using SVM

→ When we use Classification Using PCA we will get Test Misclassification Rate around 50 percent

→ When we use Classification Using MULTICLASS LO- GESTIC DISCRIMINATION we will get Test Misclassification Rate around 2.5 percent

→ When we use Classification Using SVM we will get Test Misclassification Rate around 54.2 percent when kernel is ”Polynomial” and around 13.2 percent when kernel is ”Radial”

→ As MULTICLASS LOGESTIC DISCRIMINATION method gives minimum misclassification rate so we have use it for predicting class level for original Test set in which response variable (Price Range) is absent

→

MULTICLASS LOGESTIC DISCRIMINATION model

predict that about 24.9 percent observation in class”0”,about 23.4percentobservatinclass”1”,about25.6percentobservation in class ”2”,about 26.1 percent observat in class ”3” in test set.

References:-

(1) An Introduction to Statistical Learning with Appli- cations in R by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani.

(2) The Elements of Statistical Learning:Data Mining, Inference, and Prediction.

(3) www.analyticsvidhya.com

(4) stats.stackexchange.com

(5) datasciencebeginners.com

(6) setscholars.net

23

R code: -

```
#install.packages("mice")
library(mice)
#install.packages("dummies")
library(dummies)
setwd("C:/Users/hp/OneDrive/Documents/R")
train_data=read.csv("data mobile price prediction_train.csv")
test_data=read.csv("data mobile price prediction_test.csv")
# to seeing the data set
colnames(train_data)
colnames(test_data)     # clearly price range is response variable
test_data=test_data[,-1]
test_data$price_range=1    # add a column
combi=rbind(train_data,test_data)
# searching for missing value
md.pattern(combi)    # no missing value find
#remove dependent and identifier variable
my_data=subset(combi,select = -c(price_range))
colnames(my_data)
str(my_data)
# There is no need of changing catogorical column into numeric as all predictors are numeric
# devide the data into training set and test set
train=my_data[1:nrow(train_data),]
test=my_data[(nrow(train_data)+1):(nrow(train_data)+nrow(test_data)),]
cor(train)   # as correlation between predictors is very less so we can not remove any variable
in this stage

#--------------------------------------------------------dealing with PC--------------------------------------------
-----------------------------


#The base R function prcomp() is used to perform PCA. By default, it centers the variable to
have mean equals to zero.
#With parameter scale. = T, we normalize the variables to have standard deviation equals to 1.
pc=prcomp(train,scale. = T)
names(pc)
pc$center  # mean of var
pc$scale  # SD of var
pc_load=pc$rotation  # loadins of pc
pc$sdev    # SD of pc
pc_var=(pc$sdev)^2
mat=matrix(rep(0,2*ncol(my_data)),ncol = 2)
mat[1,1]="no of pc"
mat[1,2]="cumulative proportion var explained"
for (i in 2:ncol(my_data)){
  mat[i,1]=i-1
  mat[i,2]=100*sum(pc_var[1:i])/sum(pc_var)
}
```

```
mat
total=0
count=1
for (i in 2:44){
  total=mat[i,2]
  if (total>80){
    j=i-1
    break
  }
}
sprintf("first %i pc explain 80 percent variation  ", j)
biplot(pc,scale=0)#The parameter scale = 0 ensures that arrows are scaled to represent the loadings.
prop_var=pc_var/sum(pc_var)
# scree plot
plot(prop_var,xlab="principal component",ylab = "proportion of var",type="b",main = "scree plot")
# cumulative scree plot (CSP)
CSP=mat[2:nrow(mat),2]
CSP
plot(CSP,xlab = "principal component",ylab = "cumulative proportion of variance",type="b",main="cumulative scree plot")


#------------------------------------------------------k mean clustering---------------------------------------------------------------------------------------

# k mean clustering
#install.packages("ggfortify")
library(ggfortify)
par(mfrow=c(1,2))
autoplot(pc,loadings=T,loadings.lable=T,main="Scatter plot between PC1 and PC2 without labels")
autoplot(pc,loadings=T,loadings.lable=T,shape=F,main="Scatter plot between PC1 and PC2 with labels")
# removing outliers
train=train[-c(1407,1694,1772),]
# now we will apply KNN to new trainig set
# from scatter diagrame no of cluster is not clear as points are closely aggregated
set.seed(2)
kmean1=kmeans(train,3,nstart=40)    # no of cluster is three
names(kmean1)
total=kmean1$totss
kmean1$withinss
within=kmean1$tot.withinss
between=kmean1$betweenss
kmean1$size
within_prop=(within/total)*100
bet_prop=(between/total)*100
within_prop
bet_prop
```

```
# when nstart=1
kmean1=kmeans(train,3,nstart=1)    # no of cluster is three
names(kmean1)
total=kmean1$totss
kmean1$withinss
within=kmean1$tot.withinss
between=kmean1$betweenss
kmean1$size
within_prop=(within/total)*100
bet_prop=(between/total)*100
within_prop
bet_prop

pc1=prcomp(train,scale. = T)
Y=pc$x[,1:2]
kmean2=kmeans(Y,3,nstart = 40)
kmean2
par(mfrow=c(1,1))
plot(Y,col=(kmean2$cluster+1),main="visualisation of data using KNN(k=3) by PC1 and PC2")
total2=kmean2$totss
kmean2$withinss
within2=kmean2$tot.withinss
between2=kmean2$betweenss
kmean2$size
within_prop2=(within2/total2)*100
bet_prop2=(between2/total2)*100
within_prop2
bet_prop2

#-------------------------------------------------------CLASSIFICATION--------------------------------------------
----------------------
#--------------------------------------------------CLASSIFICATION USING PCA-----------------------------------
---------------

#install.packages("rpart")
library(rpart)
#add a training set with principal components
train.data <- data.frame(price_range = train_data$price_range, pc$x)
#we are interested in first j PCAs
k=j+1
train.data <- train.data[,1:k]   # as 1st column is price range in train.data
# partion of training set into 3:1
n=nrow(train.data)
n1=n*3/4
n2=n*1/4
error1=rep(0,n2)
# when the training set has size 1500 and test set has 500
for(i in 1:n2){
  train1=train.data[i:(n1+i-1),]
  test1=train.data[-(i:(n1+i-1)),]
```

```
  model=rpart(price_range ~ .,data = train.data, method = "anova")
  Y_cap=predict(model,newdata=test1)
  Y_cap=round(Y_cap)
  Y=subset(test1,select=c(price_range))
  Y=Y[,1]

  k1=0
  for(j1 in 1:n2){
    if(Y_cap[j1]!=Y[j1]){
      k1=k1+1
    }
  }
  error1[i]=(1/n2)*k1
}
test_error_PCA1=mean(error1)

# when the training set has size 500 and test set has 1500
error2=rep(0,n2)
for(i in 1:n2){
  train1=train.data[i:(n2+i-1),]
  test1=train.data[-(i:(n2+i-1)),]
  model=rpart(price_range ~ .,data = train.data, method = "anova")
  Y_cap=predict(model,newdata=test1)
  Y_cap=round(Y_cap)
  Y=subset(test1,select=c(price_range))
  Y=Y[,1]

  k2=0
  for(j2 in 1:n1){
    if(Y_cap[j2]!=Y[j2]){
      k2=k2+1
    }
  }
  error2[i]=(1/n1)*k2
}
test_error_PCA2=mean(error2)
mat1=matrix(rep(0,4),ncol = 2,byrow=T)
mat1[1,1]="missclassification test error when training set size is 1500"
mat1[1,2]="missclassification test error when training set size is 500"
mat1[2,1]=test_error_PCA1
mat1[2,2]=test_error_PCA2
mat1
test_error_PCA=test_error_PCA1
test_error_PCA

#---------------------------------------------MULTICLASS LOGESTIC DISCRIMINATION------------------------
------------------------------------------------
#install.packages("nnet")
library(nnet)
setwd("C:/Users/hp/OneDrive/Documents/R")
data_train=read.csv("data mobile price prediction_train.csv")
```

```
data_test=read.csv("data mobile price prediction_test.csv")
# now first we use multinomial regression with some of independent predictor (probabilly which affects response more)
# now we will build logestic model on entire data set. after removing response var
model2=multinom(price_range~.,data = data_train)
k2=summary(model2)
# now we will comput valaue using Z score
z= k2$coefficients/k2$standard.errors
z
p_val=2*(pnorm(abs(z),lower.tail = F))
p_val   # clearly there are manr pridictors that are significant
print("from p value clearly of blue,fc,pc, sc_h,sc_w,talk_time are not significant")
# removal of unsignificant and identity variable from train and test data.
data_test=subset(data_test,select = -c(id))     # removal of identity var
data_test$price_range=1    # add a column to test data
combine_data=rbind(data_train,data_test)
colnames(data_test)
colnames(data_train)
combine_data=subset(combine_data,select=-c(blue,fc,pc, sc_h,sc_w,talk_time))
ncol(combine_data)
train=combine_data[1:nrow(data_train),]
test=combine_data[-(1:nrow(data_train)),]
# Now we apply different tecnique. First split training set into 3:1 in which bigger portion is considered as taraining set and
# smaller portion is considerd as test as. and then will calculate missclassification rate on this trainig set.And then use best method
# in this scenario for class prediction of real test set in which response is not given

n1=nrow(train)*(3/4)   # n1 is size of trainig set
n2=nrow(train)*(1/4)   # n2 is size of test set
# size of trainig set is 1500 1nd test set is 500
error1=rep(0,n2)
for(i in 1:n2){
  train1=train[i:(n1+i-1),]
  test1=train[-(i:(n1+i-1)),]
  model=multinom(price_range~.,data=train1)
  Y_cap=predict(model,newdata=test1)
  Y=subset(test1,select=c(price_range))
  Y=Y[,1]
  k=0
  for(j in 1:n2){
    if(Y_cap[j]!=Y[j]){
      k=k+1
    }
  }
  error1[i]=(1/n2)*k
}
test_error_log1=mean(error1)

# size of trainig set is 500 1nd test set is 1500
error2=rep(0,n2)
```

```
for(i in 1:n2){
  train1=train[i:(n2+i-1),]
  test1=train[-(i:(n2+i-1)),]
  model=multinom(price_range~.,data=train1)
  Y_cap=predict(model,newdata=test1)
  Y=subset(test1,select=c(price_range))
  Y=Y[,1]
  k=0
  for(j in 1:n2){
    if(Y_cap[j]!=Y[j]){
      k=k+1
    }
  }
  error2[i]=(1/n2)*k
}
test_error_log2=mean(error2)
mat1=matrix(rep(0,4),ncol = 2,byrow=T)
mat1[1,1]="missclassification error when training set size is 1500"
mat1[1,2]="missclassification error when training set size is 500"
mat1[2,1]=test_error_log1
mat1[2,2]=test_error_log2
mat1
test_error_log=test_error_log1
test_error_log

#------------------------------------------------------------SUPPORT VECTOR MACHINE-----------------------
-----------------
#install.packages("e1071")
library(e1071)
setwd("C:/Users/hp/OneDrive/Documents/R")
data=read.csv("data mobile price prediction_train.csv",,header = T)
# converting data into 3:1 for training and test
n=nrow(data)
n1=n*3/4
n2=n*1/4
set.seed(1)
rand_sam=sample(n,n1)
# as we know that data is very aggrigated so linear classifier will not be suitable tecnique so we
will use polynomial and radial as kernel
train=data[rand_sam,]
test=data[-rand_sam,]
train_x=train[,1:20]
train_y=train[,21]
test_x=test[,1:20]
test_y=test[,21]
dat=data.frame(x=train_x,y=as.factor(train_y))
test_data=data.frame(x=test_x,y=as.factor(test_y))
#when kernal is polynomial
svmfit_poly=svm(y~.,data=dat,kernel="polynomial",ranges=list(cost=c(0.1,1,10)),degree=c(4,6,
8))
tune_poly=tune(svm,y~.,data=dat,kernel="polynomial",ranges=list(cost=c(0.1,1,10)),degree=c(
```

```
4,6,8))
bestmod_poly=tune_poly$best.model
summary(bestmod_poly)
ypred_poly=predict(bestmod_poly,test_data)
tab1=table(predict=ypred_poly,true=test_y)
tab1
test_error_svm_poly=1-(sum(diag(tab1)))/sum(tab1)
test_error_svm_poly
#when kernal is radial
svmfit_rad=svm(y~.,data=dat,kernel="radial",ranges=list(cost=c(0.1,1,10)),gamma=c(4,6,8))
tune_rad=tune(svm,y~.,data=dat,kernel="radial",ranges=list(cost=c(0.1,1,10)),gamma=c(4,6,8)
)
bestmod_rad=tune_rad$best.model
summary(bestmod_rad)
ypred_rad=predict(bestmod_rad,test_data)
tab2=table(predict=ypred_rad,true=test_y)
tab2
test_error_svm_rad=1-(sum(diag(tab2)))/sum(tab2)
test_error_svm_rad

#------------------------------------------comparision of Test_error of different method------------------
---------------------
x=c("PCA","LOGESTIC","SVM WITH POLYNOMIAL KERNEL","SVM WITH RADIAL KERNEL")x
x=c(1,2,3,4)
y=c(test_error_PCA,test_error_log,test_error_svm_poly,test_error_svm_rad)
plot(x,y,type ="b",sub="1=PCA  2=LOGESTIC  3=SVM WITH POLYNOMIAL KERNEL  4=SVM WITH
RADIAL KERNEL",cex=2,
   xlab="name of method indicating bu no",ylab = " amoount of test error",main="TEST
ERROR",col="red")

#-------------------------------------------classification of test data by best model--------------------------
-------------
#install.packages("nnet")
library(nnet)
setwd("C:/Users/hp/OneDrive/Documents/R")
train_data=read.csv("data mobile price prediction_train.csv")
test_data=read.csv("data mobile price prediction_test.csv")
# as test error is minimum for MULTICLASS LOGESTIC DISCRIMINATION so we predict classes
for test data using it.
fit=multinom(price_range~.,data=train_data)
ypredict=predict(fit,newdata = test_data)
test_summary=summary(ypredict)
test_summary
test_class_prop=test_summary/1000
test_class_prop
train_resp=train_data[,21]
train_summary=table(train_resp)
train_summary
train_class_prop=train_summary/2000
rbind(train_class_prop,test_class_prop)
```