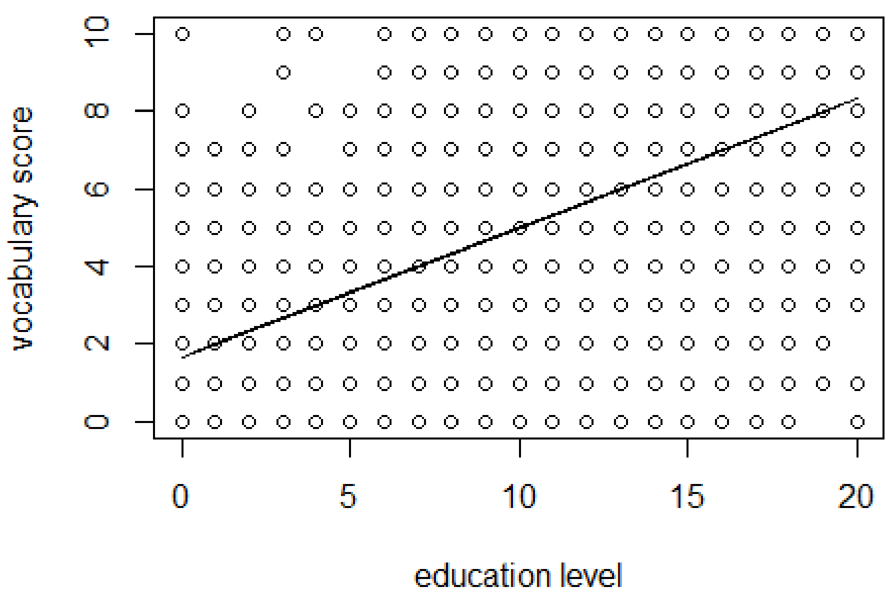
B565_hw5

1

```
# (a)

## Reading the dataframe and subsetting column 'education' and 'vocabulary'
df = read.csv("D:/Data Mining/hw5/Vocab.csv")
df1 = df[,c("education","vocabulary")]

## Creating feature vector X and response vector Y
X1 = df1[,1]
X2 = rep(1,length(X1))
X = cbind(X1,X2)
Y = df1[,2]

head(X)

##      X1 X2
## [1,] 14  1
## [2,] 16  1
## [3,] 10  1
## [4,] 10  1
## [5,] 12  1
## [6,] 16  1

# (b)

## Solving for w and reporting values of a and b
w = solve(t(X)%*%X)%*%(t(X)%*%Y)
a = w[1]
b = w[2]

a

## [1] 0.3318736

b

## [1] 1.677939

## predicting yhat from w
yhat = X%*%w
df1$pred_vocab = yhat
head(df1)

##   education vocabulary pred_vocab
## 1        14          9   6.324170
```

```
## 2         16          9   6.987917
## 3         10          9   4.996675
## 4         10          5   4.996675
## 5         12          8   5.660423
## 6         16          8   6.987917

## plot fiited line for prediction
plot(X1,Y, xlab = "education level", ylab="vocabulary score")
lines(X1,yhat)
```



(c) As seen from the plot, increase in education level has higher vocabulary score. So, yes people with more education tend to have larger vocabularies from predicted values.

(d) Since we got coefficient of X i.e. year of education as positive with a value of 0.33, so with unit increase in year of education, there will be 0.33 unit increase in voculabulary score on average.

2

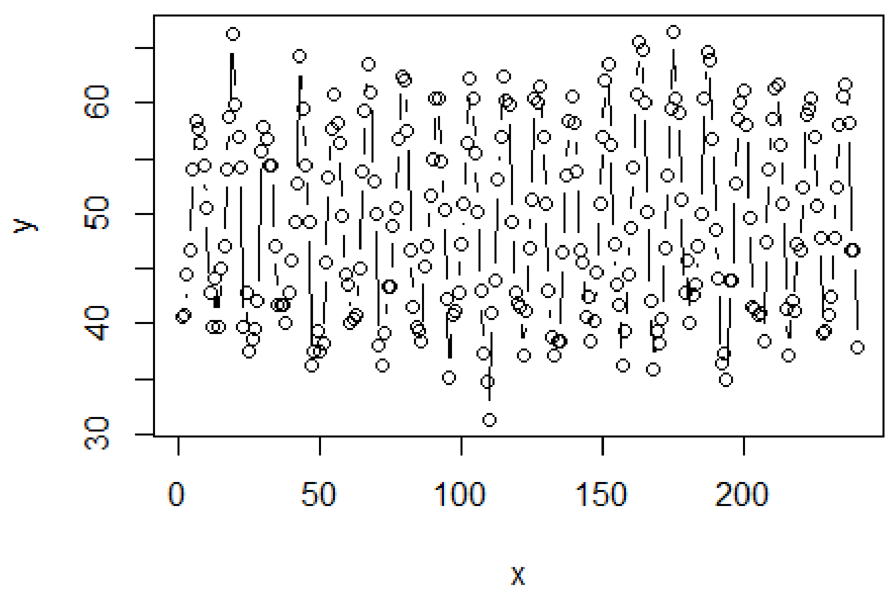
```
# (a)

## Reading the dataframe and creating feature vector X and reponse vector Y
df = read.csv("D:/Data Mining/hw5/ais.csv",stringsAsFactors=FALSE,sep=",")

X1 = df[,c(3,4,5,6,7,8,9,10,11,12)]
X2 = rep(1,nrow(X1))
```

```r
X = cbind(X1,X2)
X = as.matrix(X)
Y = df[,2]

## Solving for w
w = solve(t(X)%*%X)%*%t(X)%*%Y
w

##                  [,1]
## wcc     1.131456e-03
## hc      1.047225e-01
## hg      3.266434e-02
## ferr    3.230767e-05
## bmi    -2.535350e-02
## ssf     3.472695e-03
## pcBfat -9.302996e-03
## lbm     9.284090e-03
## ht     -4.449059e-03
## wt     -2.426072e-03
## X2      5.685820e-01

# (b)

## Computing error using predicted and actual values and summing up square of
error to get sse
yhat = X%*%w
error = (Y-yhat)^2

sse = sum(error)
sse

## [1] 5.905161

# (c)

## Creating sum of square error by removing each feature from X matrix
sse1 = rep(0,11)
for (i in 1:ncol(X)){
X1 = X[,-c(i)]
w1 = solve(t(X1)%*%X1)%*%t(X1)%*%Y
yhat1 = X1%*%w1
error1 = (Y-yhat1)^2

sse1[i] = sum(error1)
}
sse1

##  [1] 5.905920 8.517387 5.937395 5.905548 5.920589 6.019063 5.915726
##  [8] 5.911871 5.913026 5.905610 5.909294
```

As from the sse1, we can see that variable ‘hc’ ommision causes the greatest increase in sse. So, omitting ‘hc’ from X is causing the greatest harm. Hence, ‘hc’ variable is most important in predicting rcc.

3

```
## Reading the data and plotting monthly beer sales
data(nottem)
y = nottem
n = length(y)
x = 1:n

# (a)

plot(x,y,type="b")
```




```
# (b)

## Creating feature vectors X using cos and sign function
x1 = cos(2*pi*x/12)
x2 = sin(2*pi*x/12)
x3 = rep(1,n)

X = cbind(x1,x2,x3)

## Solving for w  and reporting values of a, b and c
```

```r
w = solve(t(X)%*%X)%*%t(X)%*%y
a = w[1]
b = w[2]
c = w[3]
a
```

```
## [1] -9.240921
```

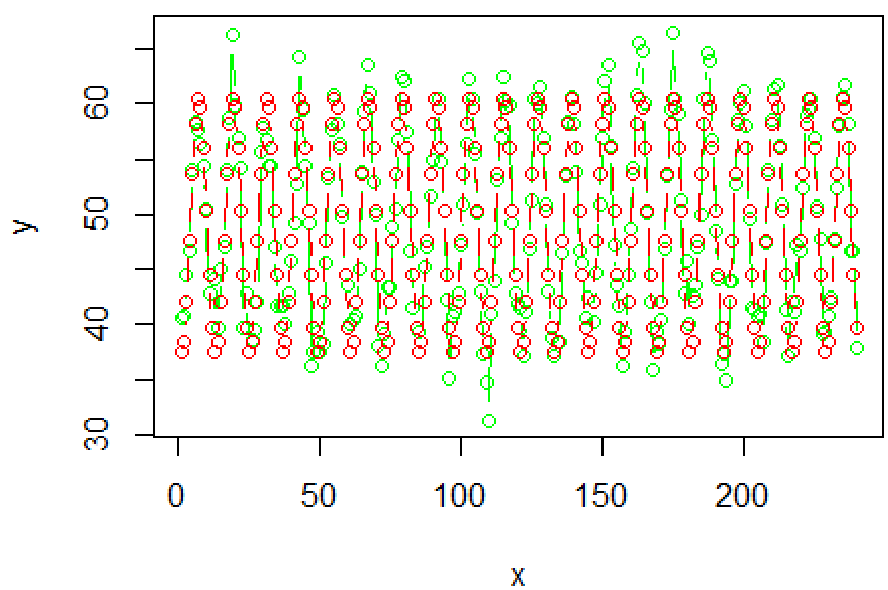
```r
b
```

```
## [1] -6.940906
```

```r
c
```

```
## [1] 49.03958
```

```r
## Predicting yhat from w
yhat = X%*%w

## plotting actual and predicted values of monthly beer sales
plot(x,y,type="b",col="green")
lines(x,yhat,type="b",col="red")
```

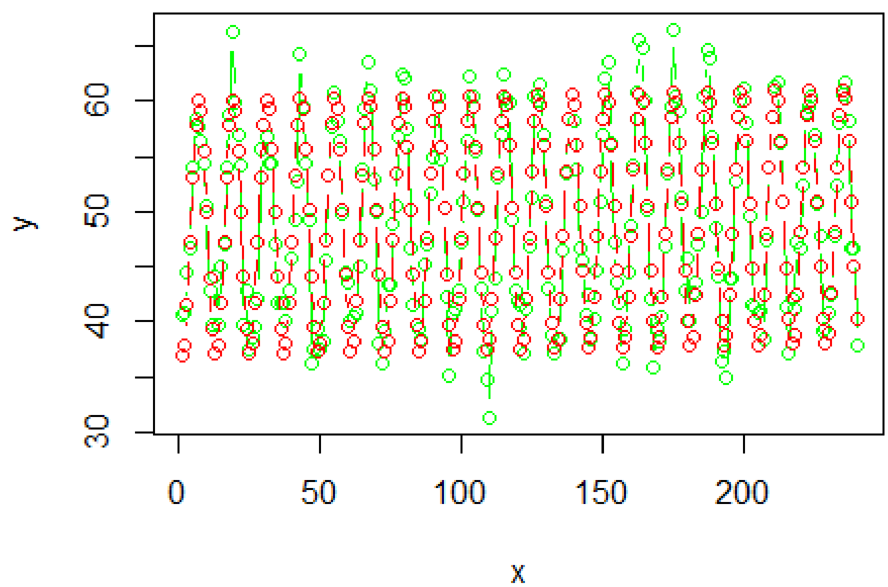



```r
# (c)

## Adding X in the feature vector
X1 = cbind(x,x1,x2,x3)
```

```r
## Solving for w using new feature vector
w1 = solve(t(X1)%*%X1)%*%t(X1)%*%y
w1
```

```
##            [,1]
## x   0.004392239
## x1 -9.245313509
## x2 -6.924513489
## x3 48.510318555
```

```r
## Predicting yhat1 from new w
yhat1 = X1%*%w1

## Plotiing actual and
plot(x,y,type="b",col="green")
lines(x,yhat1,type="b",col="red")
```



From the plot, we can see the trend as positive. Also, since the coeficient is positive for x, so with increase in month (x), the company is experiencing growth of 0.4% in sales.

4

```r
# (a)

## Reading feature vector 1 and 2 and response vector 1 and 2
## Splitting into train and test using first and second half of dataset
```

```
x1 = as.matrix(read.table("D:/Data Mining/hw5/pred1.dat"))
y1 = as.matrix(read.table("D:/Data Mining/hw5/resp1.dat"))

x2 = as.matrix(read.table("D:/Data Mining/hw5/pred2.dat"))
y2 = as.matrix(read.table("D:/Data Mining/hw5/resp2.dat"))

x1_train = x1[1:(nrow(x1)/2),]
y1_train = y1[1:(nrow(x1)/2),]

x1_test = x1[((nrow(x1)/2)+1):nrow(x1),]
y1_test = y1[((nrow(x1)/2)+1):nrow(x2),]


## Solving for w for x1_train
w1 = solve(t(x1_train)%*%x1_train)%*%t(x1_train)%*%y1_train
w1

##              [,1]
## V1  -8.375191e-05
## V2   2.295239e-03
## V3  -1.041287e-02
## V4  -3.820085e-03
## V5   6.505296e-03
## V6  -3.067989e-03
## V7   8.297986e-03
## V8   4.166908e-03
## V9  -2.002319e-04
## V10  3.821133e-03
## V11 -5.333165e-03
## V12  3.929818e-03
## V13 -1.140321e-03
## V14  2.505379e-03
## V15 -4.060783e-03
## V16 -2.997473e+00
## V17  2.791958e-03
## V18 -4.740269e-03
## V19  1.371139e-03
## V20  1.588981e-03
## V21  1.000069e+01
## V22  4.924661e-03
## V23  4.224946e-03
## V24  4.613569e-03
## V25 -4.775163e-03
## V26 -1.981545e-03
## V27  7.744521e-03
## V28 -1.963880e-04
## V29 -3.143477e-03
## V30  8.495242e-03
```

```
## V31 -7.979740e-04
## V32 -1.180831e-02
## V33  5.426022e-03
## V34 -4.268450e-03
## V35 -1.365428e-03
## V36 -3.838473e-03
## V37  2.367878e-03
## V38 -2.711016e-03
## V39  2.289859e-03
## V40  1.269390e-03
## V41 -2.503667e-03
## V42  4.722254e-03
## V43 -6.893779e-04
## V44  4.504424e-03
## V45  3.782918e-03
## V46  3.775611e-03
## V47  4.997276e+00
## V48  3.653995e-03
## V49 -7.932763e-03
## V50  1.706155e-03
```

```
x2_train = x2[1:(nrow(x2)/2),]
y2_train = y2[1:(nrow(x2)/2),]

x2_test = x2[((nrow(x2)/2)+1):nrow(x2),]
y2_test = y2[((nrow(x2)/2)+1):nrow(x2),]

## Solving for w for x2_train
w2 = solve(t(x2_train)%*%x2_train)%*%t(x2_train)%*%y2_train
w2
```

```
##              [,1]
## V1    10.90739810
## V2   -30.27403868
## V3    21.93560645
## V4    -4.28957288
## V5     7.84720442
## V6    10.20155807
## V7   -20.12190301
## V8    -2.85715860
## V9     9.34188425
## V10  -25.18681608
## V11    8.59363245
## V12  -18.68309085
## V13   -9.64635252
## V14    5.46811679
## V15   13.72070914
## V16   20.06464420
## V17   -8.83395654
## V18    4.73872502
```

```
## V19   -3.61601614
## V20   -4.78403063
## V21   -4.77130486
## V22  -20.50199289
## V23   27.31445241
## V24    0.36499886
## V25    5.13917748
## V26   -4.52897374
## V27   -2.29204041
## V28  -12.11356719
## V29  -17.95829473
## V30   -0.32911195
## V31   -9.70094972
## V32   -5.28577219
## V33   21.18217261
## V34  -18.77152554
## V35   12.90975101
## V36    3.40124088
## V37  -19.49552173
## V38  -17.84471555
## V39   -9.36031883
## V40   11.40579874
## V41    0.25657045
## V42   -7.12667829
## V43   17.56888378
## V44   -0.77321587
## V45   12.18878455
## V46    8.12801031
## V47   -6.67286627
## V48   -9.91292052
## V49   15.83889826
## V50    5.21562771
## V51    6.24882579
## V52   -9.79070960
## V53   -9.16312209
## V54  -23.34539501
## V55    3.42039505
## V56  -11.41000869
## V57   25.98515166
## V58    2.25640680
## V59    3.83827712
## V60  -12.63025944
## V61  -14.18214791
## V62    3.18614808
## V63    0.41994018
## V64   10.50244739
## V65   -3.70638281
## V66   16.82186312
## V67  -11.14373713
## V68    0.01437147
```

```
## V69    4.81792526
## V70   16.89008185
## V71   -8.39891094
## V72    0.85731853
## V73   -8.55724804
## V74   -6.43050556
## V75    6.76253557
## V76    5.44829681
## V77   14.38341039
## V78  -14.55701675
## V79  -10.56293910
## V80    4.04129746
## V81    6.64964591
## V82   11.70630210
## V83   -0.18059933
## V84   21.67505418
## V85   11.81585951
## V86   16.37496047
## V87   23.25763233
## V88  -18.36036837
## V89   -0.67409762
## V90   25.61518082
## V91   15.16094138
## V92    0.68993781
## V93   -9.70150291
## V94    4.98827046
## V95   12.04171047
## V96   11.69067930
## V97   -3.63645822
## V98  -15.30890661
## V99   15.33608222
## V100   1.05364403
## V101  13.00108743
## V102  -0.67967520
## V103 -15.39022082
## V104  -6.17605007
## V105  -0.97105323
## V106   8.80337889
## V107  -9.94534654
## V108  11.32104569
## V109  -0.53891335
## V110   5.36995199
## V111   9.45576382
## V112  24.71778632
## V113  -9.52011239
## V114   3.59691754
## V115   1.84436591
## V116  -4.50937931
## V117   1.44185158
## V118 -10.40389727
```

```
## V119  12.73475218
## V120   3.58479427
## V121  -7.37404057
## V122   9.40610715
## V123 -17.56886772
## V124  -6.26004560
## V125  12.59014213
## V126  14.33751286
## V127  22.19235866
## V128   8.87655672
## V129  -3.77696261
## V130   4.55218154
## V131  -3.63947346
## V132 -19.82853929
## V133  14.67832776
## V134   9.91763182
## V135   0.68782246
## V136  -4.23264075
## V137 -18.70424105
## V138  10.01653994
## V139  -8.75719632
## V140   9.34039042
## V141   2.30883668
## V142  -3.06337182
## V143  -2.12724139
## V144  -6.90498676
## V145  -2.37057193
## V146   0.53181835
## V147  -1.90465128
## V148 -14.52251670
## V149  14.64502815
## V150  -6.93854550
## V151  18.40803710
## V152   1.11545992
## V153 -20.89396397
## V154  13.37879205
## V155  18.04254166
## V156  14.01272830
## V157   1.34105715
## V158   9.62988461
## V159  -5.27985610
## V160  -6.53990466
## V161   6.51949269
## V162 -11.43437021
## V163  -2.89160540
## V164  -3.69499047
## V165  -6.35988493
## V166   5.23787312
## V167  -4.96727716
## V168  -1.93996759
```

```
## V169  -8.06650640
## V170  -3.28332573
## V171 -15.78039642
## V172   9.96692136
## V173  -1.92771743
## V174 -16.75882813
## V175   3.84458687
## V176  -8.50413587
## V177 -17.18660604
## V178   2.53919867
## V179  17.95502165
## V180  24.64641047
## V181   1.61473022
## V182   2.62031236
## V183  -1.59809068
## V184  -3.99169411
## V185   6.34690447
## V186   7.25824879
## V187   6.85366657
## V188 -21.24378897
## V189  16.00916343
## V190  -5.86383039
## V191   6.32714642
## V192 -10.79146354
## V193 -11.07176334
## V194   1.21749552
## V195   1.18727930
## V196  -3.13676023
## V197  16.01742046
## V198   5.53730820
## V199  14.42076038
## V200   8.09668082
## V201  -1.15397849
## V202  10.64666608
## V203 -19.10603306
## V204  11.42983986
## V205  -2.66617117
## V206  -4.90423552
## V207   3.35684941
## V208  12.32028881
## V209  -3.52519423
## V210 -13.87204628
## V211 -10.63283852
## V212 -18.18539838
## V213   2.03519195
## V214   4.42430318
## V215   1.53021900
## V216   5.05034710
## V217 -11.15832887
## V218  11.77198788
```

```
## V219  -8.67666185
## V220   3.17494042
## V221   1.97468867
## V222 -18.69323077
## V223 -14.39909841
## V224  -4.02204836
## V225  -9.54734089
## V226   0.41051888
## V227 -10.58682830
## V228 -15.52804754
## V229   2.56049032
## V230  16.73631332
## V231 -10.66768293
## V232   1.50411321
## V233   2.86324480
## V234   5.64988687
## V235   8.26458818
## V236 -11.86734832
## V237  11.65037205
## V238  -7.98930946
## V239   4.20715077
## V240  11.18752749
## V241  25.07107929
## V242 -19.68816166
## V243  -1.25329326
## V244  -1.27830939
## V245  -4.88760962
## V246  -0.15806890
## V247  10.47110583
## V248 -10.67435167
## V249   5.26470836
## V250   8.54931381
## V251 -13.94984322
## V252   9.95568483
## V253  -4.91340789
## V254  -1.71088707
## V255   8.73764873
## V256  11.90328252
## V257   4.24304252
## V258  -4.80869093
## V259  -9.97317463
## V260  -9.55657900
## V261  -8.44435164
## V262   4.51805454
## V263  -8.23426200
## V264  22.62114344
## V265 -12.72811725
## V266  -6.22553991
## V267  -0.43046759
## V268  -9.18548835
```

```
## V269   3.62973687
## V270   2.65891894
## V271   3.54529818
## V272  18.98510702
## V273   2.20899294
## V274   7.76078534
## V275 -11.32754380
## V276  -5.56261478
## V277 -16.47497730
## V278   6.91640558
## V279 -11.54622043
## V280 -15.48702288
## V281  -0.33631507
## V282  14.70113483
## V283 -22.23776568
## V284   1.01018834
## V285  -6.54749136
## V286   0.42430456
## V287   0.17884745
## V288   0.57683359
## V289   3.64088137
## V290  -0.45373813
## V291   3.84290546
## V292   1.16827651
## V293 -10.95444912
## V294  -0.28732044
## V295   2.61339112
## V296   5.54141933
## V297   8.23383722
## V298 -19.26134719
## V299   3.55638218
## V300   3.16473285
## V301  -6.41931706
## V302 -18.55072693
## V303   7.67539417
## V304   7.26798807
## V305 -11.75484527
## V306 -13.93329115
## V307   7.31267133
## V308  -0.61404312
## V309   1.05293944
## V310  -8.67219226
## V311  -3.03346040
## V312   9.26375969
## V313  24.46995318
## V314   0.08222980
## V315   1.35514001
## V316 -13.20321269
## V317   8.28265939
## V318  28.78186959
```

```
## V319   4.39512215
## V320 -15.96207564
## V321 -25.47452491
## V322  14.80109835
## V323  11.27876561
## V324  -0.77548478
## V325   5.31493221
## V326  -2.77344754
## V327  -5.00410091
## V328   2.13458205
## V329  -1.77263980
## V330  12.16775164
## V331  -7.07653720
## V332  14.33624449
## V333  16.88267208
## V334   9.61443361
## V335  -4.55424569
## V336  -2.29776740
## V337  -0.37622021
## V338  -4.54476975
## V339   1.97420741
## V340  -6.26633898
## V341   9.93159169
## V342  -1.11633505
## V343 -22.75057253
## V344  14.31558436
## V345   1.11763600
## V346  29.36587972
## V347 -10.72432030
## V348   6.94570871
## V349   3.22187663
## V350   6.22710428
## V351  14.96284728
## V352  -1.20081082
## V353 -15.77756988
## V354  -8.81092658
## V355   3.49878738
## V356  -3.79627174
## V357  -9.54846025
## V358   6.01122540
## V359   2.26747104
## V360 -19.24950818
## V361 -11.38426336
## V362  -5.85304620
## V363  14.20827557
## V364  13.97458140
## V365  -6.02570766
## V366   2.98879225
## V367  -2.30184031
## V368  -7.48987967
```

```
## V369  14.05220411
## V370   5.71910379
## V371  -3.52635946
## V372  22.25104721
## V373   2.80920116
## V374   0.92230586
## V375 -29.35423684
## V376   9.00144922
## V377  12.77867948
## V378  -6.41164265
## V379  -1.23612194
## V380  29.14504670
## V381  10.95677756
## V382 -17.56878840
## V383  -0.50800569
## V384  -8.16616525
## V385  -6.23204956
## V386  -2.35117670
## V387  -5.01210648
## V388  -2.84633579
## V389   1.13418508
## V390   1.94191526
## V391  -7.52251294
## V392  -8.89176010
## V393 -15.94538577
## V394   3.74234144
## V395   0.47854164
## V396 -13.96735409
## V397   4.60782559
## V398 -15.06195574
## V399  15.23401727
## V400  12.82988566
## V401 -10.45310517
## V402  19.92497754
## V403  -5.82824763
## V404 -33.07652103
## V405  -1.59740990
## V406   7.33967347
## V407   1.91628281
## V408 -16.78216628
## V409  11.41822831
## V410  -5.75558675
## V411  -7.58977135
## V412  -8.40601600
## V413   6.38783535
## V414 -12.22059574
## V415   6.93586169
## V416 -23.94239376
## V417  -6.45892645
## V418  18.71964170
```

```
## V419   6.79807513
## V420   3.47961152
## V421   3.17915866
## V422   7.17520463
## V423 -10.50612382
## V424  -6.63927046
## V425   2.23806015
## V426  -8.74337116
## V427  12.55127113
## V428  -9.63089987
## V429   6.31312102
## V430 -13.42538255
## V431 -14.91591612
## V432   1.97025073
## V433 -23.05346817
## V434   0.49614174
## V435  -5.32032179
## V436 -16.71188941
## V437   4.05279471
## V438   1.01196576
## V439  -2.27504695
## V440  -6.32993344
## V441  13.84032070
## V442   0.22653870
## V443  -6.43403445
## V444  -6.38047276
## V445  -4.73025318
## V446 -18.09848045
## V447   7.69291569
## V448 -16.17209985
## V449   8.72369425
## V450  11.91687965
## V451  16.74366411
## V452 -11.42280610
## V453 -20.18454638
## V454 -23.27338720
## V455   2.60427152
## V456  23.18796361
## V457  12.03903433
## V458   2.35715132
## V459   5.53166768
## V460   2.91945482
## V461   2.31781127
## V462 -10.43569447
## V463  13.94073638
## V464 -14.95308863
## V465 -15.43315032
## V466   4.17468377
## V467  11.05671378
## V468  -1.24538881
```

```
## V469   3.54249867
## V470  -8.17266225
## V471   1.86620388
## V472 -12.66764972
## V473   4.57817313
## V474   0.19414683
## V475 -16.74363050
## V476  -2.06353932
## V477 -14.06347304
## V478  25.45618363
## V479  23.90792303
## V480 -25.85372663
## V481  -4.36547684
## V482 -15.14691759
## V483 -17.29082967
## V484   3.14434762
## V485  -3.62518647
## V486 -11.21893586
## V487  -1.23179818
## V488  35.46866323
## V489 -14.14555414
## V490  -5.41755514
## V491   1.81776312
## V492   3.32260180
## V493   5.24846853
## V494   4.51706162
## V495  -1.24558781
## V496   0.49691728
## V497   1.34691156
## V498   4.09909232
## V499  21.34433514
## V500  -3.38213312
```

```
# (b)
## Predicting y for test set using w computed from training data
## Computing SSE on test data
yhat1_test = x1_test%*%w1
error1 = (y1_test-yhat1_test)^2
SSE1 = sum(error1)
SSE1
```

```
## [1] 5.721507
```

```
## Predicting y for test set using w computed from training data
## Computing SSE on test data
yhat2_test = x2_test%*%w2
error2 = (y2_test-yhat2_test)^2
SSE2 = sum(error2)
SSE2
```

```
## [1] 32984664
```

5

```r
# (a)

## Reading the 1st data set and spliting into test and train
x = as.matrix(read.table("D:/Data Mining/hw5/pred1.dat"))
y = as.matrix(read.table("D:/Data Mining/hw5/resp1.dat"))

x_train = x[1:(nrow(x)/2),]
y_train = y[1:(nrow(x)/2),]

x_test = x[((nrow(x)/2)+1):nrow(x),]
y_test = y[((nrow(x)/2)+1):nrow(x),]

## obtaining first best predictor variable using forward selection and
reporting sum of squared error
sse = rep(0,ncol(x_train))
for (i in 1:ncol(x_train)){
X = x_train[,c(i)]
w = solve(t(X)%*%X)%*%t(X)%*%y_train
yhat = X%*%w
error = (y_train-yhat)^2

sse[i] = sum(error)
}

v1 = which.min(sse)
sse[v1]
```

```
## [1] 17332.46
```

```r
## obtaining the second best predictor variable and reporting sum of squared
error
sse1 = rep(0,ncol(x_train))
for (i in 1:ncol(x_train)){
if (i == v1) {
sse1[i] = Inf
}
else {
X = x_train[,c(i,v1)]
w = solve(t(X)%*%X)%*%t(X)%*%y_train
yhat = X%*%w
error = (y_train-yhat)^2
sse1[i] = sum(error)
}
}

v2 = which.min(sse1)
sse1[v2]
```

```
## [1] 4581.104

## obtaining the third best predictor variable and reporting sum of squared
error
sse2 = rep(0,ncol(x_train))
for (i in 1:ncol(x_train)){
if (i == v1 | i == v2) {
sse2[i] = Inf
}
else {
X = x_train[,c(i,v1,v2)]
w = solve(t(X)%*%X)%*%t(X)%*%y_train
yhat = X%*%w
error = (y_train-yhat)^2
sse2[i] = sum(error)
}
}

v3 = which.min(sse2)
sse2[v3]

## [1] 5.30266

## Reporting best three predictor variables index
v1

## [1] 21

v2

## [1] 47

v3

## [1] 16

# 3 best predictor variables are v16,v21 and v47

# (b)

# with best predictors

x_train_new = x_train[,c(16,21,47)]
x_test_new = x_test[,c(16,21,47)]

## Computing prediction on test dataset using w_new
w_new = solve(t(x_train_new)%*%x_train_new)%*%t(x_train_new)%*%y_train
yhat_test_new = x_test_new%*%w_new

## Computing SSE on test dataset
error_new = (y_test-yhat_test_new)^2
```

```
sse_new = sum(error_new)
sse_new

## [1] 5.099969

# with all predictors

## Computing prediction on test dataset using w_new
w = solve(t(x_train)%*%x_train)%*%t(x_train)%*%y_train
yhat_test = x_test%*%w

## Computing SSE on test dataset
error = (y_test-yhat_test)^2
sse = sum(error)
sse

## [1] 5.721507
```

The approach with choosing only best variables using variable selection gives a better SSE. Choosing all variables will reduce the error in the training set but gives higher error rate in test set because of overfiiting. Only important variables should be used for prediction as it will generalize well in test set.

6

```
## Reading 2nd dataset and splitting into test and train
x = as.matrix(read.table("D:/Data Mining/hw5/pred2.dat"))
y = as.matrix(read.table("D:/Data Mining/hw5/resp2.dat"))

x_train = x[1:(nrow(x)/2),]
y_train = y[1:(nrow(x)/2),]

x_test = x[((nrow(x)/2)+1):nrow(x),]
y_test = y[((nrow(x)/2)+1):nrow(x),]


# (a)

## Intialising lambda = 20
lambda = rep(20,ncol(x))
lambda = diag(lambda)

## SOlving for what using regularization and predicting on test dataset
what = solve(t(x_train)%*%x_train+lambda)%*%t(x_train)%*%y_train


yhat_test = x_test%*%what

# (b)
```

```r
## computing SSE on test dataset using ridge regression

error = (y_test-yhat_test)^2
sse = sum(error)
sse
```

```
## [1] 35918.6
```

```r
## Solving for what using plain regression and predicting on test dataset
w = solve(t(x_train)%*%x_train)%*%t(x_train)%*%y_train

yhat_test1 = x_test%*%w

## computing SSE on test dataset using plain regression

error1 = (yhat_test1-y_test)^2
sse1 = sum(error1)
sse1
```

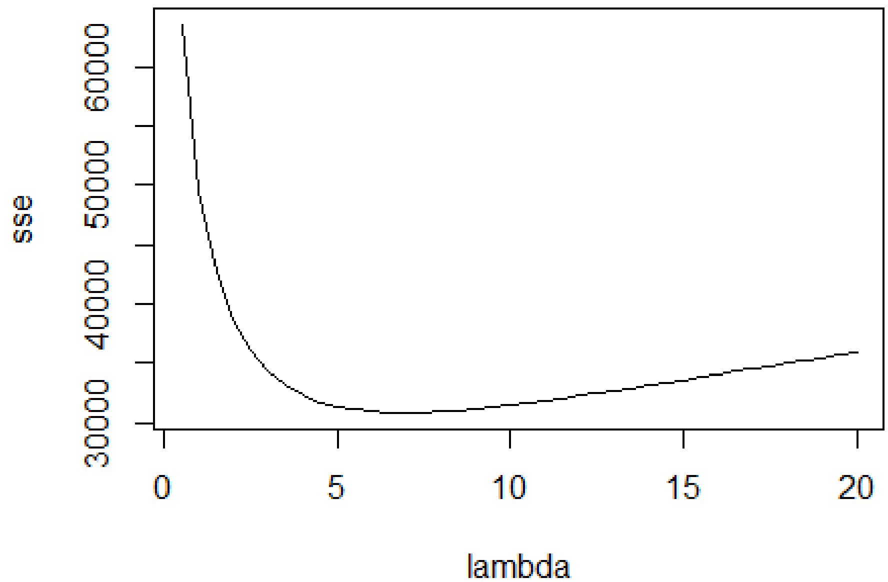
```
## [1] 32984664
```

As we can see the sum of squared error on test data is lower for ridge regression. Ridge regression is better in generalising the model because it penalises cooeficients of x so that it does not overfit. Since the cooeficients of X is penalised with lambda so the weightage to individual features will be reduced. Hence, will reduce overfitting.

6

```r
# (c)

## Using lambda from 0 to 20 with increase in 0.5
lambda = seq(0,20,by=0.5)
sse = rep(0,length(lambda))


## Computing SSE on test data set for each values of lambda
for (i in 1:length(lambda)){
lambda_matrix = diag(rep(lambda[i],ncol(x)))

what = solve(t(x_train)%*%x_train+lambda_matrix)%*%t(x_train)%*%y_train
yhat_test = x_test%*%what
error = (y_test-yhat_test)^2
sse[i] = sum(error)
}

## Plotting the SSE with lambda values as x
value = data.frame(cbind(lambda,sse))
```

```r
plot(value[2:nrow(value),],type="l")
```




```r
value[which.min(value$sse),]

##    lambda      sse
## 15      7 30788.08

# we can see minimum sse is at lambda = 7

## Using optimal lambda solve for w and compute sse on test dataset
lambda_hat = diag(rep(7,ncol(x)))
w = solve(t(x_train)%*%x_train+lambda_hat)%*%t(x_train)%*%y_train
yhat = x_test%*%w
error = (y_test-yhat)^2
sse_min = sum(error)
sse_min

## [1] 30788.08
```

7

```r
## Reading timeseries data
ts = as.matrix(read.table("D:/Data Mining/hw5/time_series.dat"))

## Creating feature vector from ts as X(i-1) and X(i-2) and reponse vector y
y = ts[3:nrow(ts),]
```

```
x1 = ts[1:(nrow(ts)-2),]
x2 = ts[2:(nrow(ts)-1),]

X = cbind(x1,x2)

## Solve for w
w = solve(t(X)%*%X)%*%(t(X)%*%y)

## Compute error using x(i) predicted and x(i) actual
yhat = X%*%w
error = y-yhat

## Reporting values of alpha1 and alpha2 which is cooeficients of X(i-1) and X(i-2) and variance of error which is e(i)
alpha1 = w[2]
alpha2 = w[1]
variance = var(error)

alpha1

## [1] 0.990185

alpha2

## [1] -0.9383054

variance

##             [,1]
## [1,] 0.002502056
```

The estimated parameters α_1 is 0.99, α_2 is -0.93 and σ^2 is 0.0025