coursework10 : Machine Learning III, Regression analysis

Fortunat Mutunda

April 12, 2016

1. Take the following data and simulate K-NN algorithm to predict the class probabilities of points (3, 5) and (4, 6). Report the probabilities with K=1, K=2 and K=3.

- I will remove the id first because it does not give any useful information, therefore it is not going to help with anything

```
str(dataf)
```

```
## 'data.frame':    10 obs. of  4 variables:
##  $ ID     : int  1 2 3 4 5 6 7 8 9 10
##  $ x_coord: int  9 2 3 4 1 3 5 6 6 3
##  $ y_coord: int  3 4 3 1 6 9 6 4 2 7
##  $ class  : int  1 1 1 1 1 0 0 0 0 0
```

```
dataf = dataf[-1]
```

Now we see that we have coordinates x and y and each point has been classified either as 1 or 0, therefore I am going to change class into factor because that's the column I am going to use to classify all the observations.

```
set.seed(9850)
gp = runif(nrow(dataf))
dataf = dataf[order(gp),]
#dataf
dataf$class = as.factor(dataf$class)
str(dataf)
```

```
## 'data.frame':    10 obs. of  3 variables:
##  $ x_coord: int  5 6 1 4 6 3 3 9 3 2
##  $ y_coord: int  6 2 6 1 4 3 7 3 9 4
##  $ class  : Factor w/ 2 levels "0","1": 1 1 2 2 1 2 1 2 1 2
```

now we can see that there is an even classification of 5 points which belong to 0 and 5 points which belong to 1.

Before I normalize the rest I am going to first randomazie everything because all the data are clissified with 5 ones then 5 zeroes I'll make it to show 0s and 1s randomly.

```
head(dataf,4)
```

```
##   x_coord y_coord class
## 7       5       6     0
## 9       6       2     0
## 5       1       6     1
## 4       4       1     1
```

Now that is done, let's proceed into normalizing the data.

```
normalize <- function(x) {
return ((x - min(x)) / (max(x) - min(x))) }
dataf_n <- as.data.frame(lapply(dataf[1:2], normalize))
```

I have just normalized x and y because class is the classifier, after doing so I can proceed into training the data

- create a training dataframe

```
#CrossTable(dataf$x_coord, dataf$y_coord)
# taining and testing on given data to see if my formula works
dataf_train = dataf_n[1:8,]
dataf_test = dataf_n[9:10,]
# then I pick the training taget which will be taken from the original dataset
dataf_train_target = dataf [1:8,3]
dataf_test_target = dataf[9:10, 3]
# done now I will try the knn to see if it works
#model1 is the trained data
model1 = knn(train = dataf_train ,test = dataf_test,cl = dataf_train_target,k = 3 )
model1
```

```
## [1] 0 1
## Levels: 0 1
```

- Now that I am sure that my trained model works I am now going to answer the question for the points (3, 5) and (4, 6) for k = 1,2,3

```
## [1] 1 0
## Levels: 0 1
```

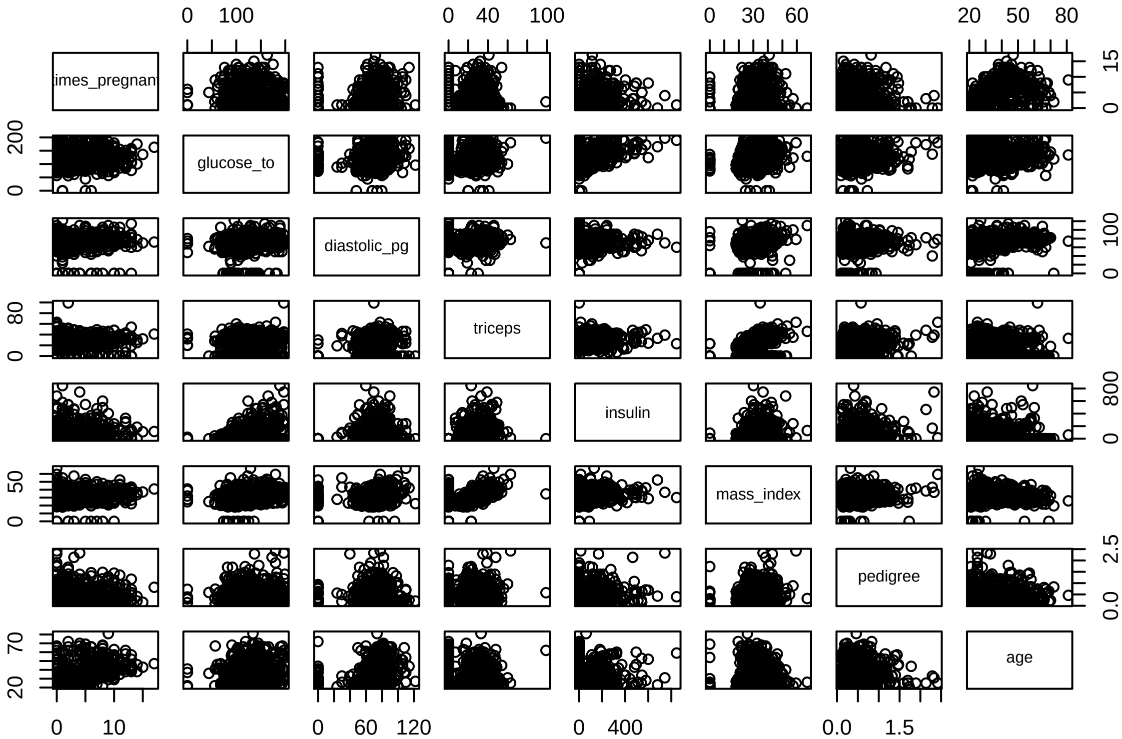
```
## [1] 1 0
## Levels: 0 1
```

```
## [1] 1 0
## Levels: 0 1
```

Conclusion: for the three measure when k =1 , 2 ,3 the result for the two given points are respectively 1-0 , 1-0 and 1-0.

2. In this task we use diabetes dataset to predict diabetes.

- Split the data randomly on 80% of training and 20% for testing.
- Fit the logistic regression on the training set to predict the class.
- Interpret the model. How the Plasma glucose concentration impacts the odds ratio of having diabetes. What about diabetes pedigree function? Which features do not affect (significantly) the risk of having diabetes?
- Now compute Accuracy, Precision, Recall and F1 score on the test set.




```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: class
##
## Terms added sequentially (first to last)
##
##
##                Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                             613     792.69
## times_pregnant  1   26.960       612     765.73 2.077e-07 ***
## glucose_to      1  132.089       611     633.64 < 2.2e-16 ***
## diastolic_pg    1    0.319       610     633.32 0.5723904
## triceps         1    2.985       609     630.33 0.0840487 .
## insulin         1    0.650       608     629.68 0.4200697
## mass_index      1   42.535       607     587.15 6.942e-11 ***
## pedigree        1   10.944       606     576.20 0.0009391 ***
## age             1    0.834       605     575.37 0.3609844
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The stars after the number can tell us which and which quality are fitted to make our classification

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: class
```

```
##
## Terms added sequentially (first to last)
##
##
##                Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                            613     792.69
## times_pregnant  1   26.960       612     765.73 2.077e-07 ***
## glucose_to      1  132.089       611     633.64 < 2.2e-16 ***
## mass_index      1   42.367       610     591.27 7.564e-11 ***
## pedigree        1    9.998       609     581.27  0.001567 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As we can see the only datas that are valuable to predict are the one remaining namely times_pregnant glucose_to and pedigree. The rest were useles to our classification of the data.

- use the fitted model to do prediction for the test data

```r
#prediction  = predict(diabetes_model,newdata = diabetes_testing)
#table(prediction)

fitted.results <- predict(diabetes_model,diabetes_testing,type='response')
fitted.results <- ifelse(fitted.results > 0.5,1,0)
misClasificError <- mean(fitted.results != diabetes_testing$class)
print(paste('Accuracy',1-misClasificError))
```

```
## [1] "Accuracy 0.779220779220779"
```

```r
table(fitted.results)
```

```
## fitted.results
##   0   1
## 113  41
```

Now let us find the confusion table

```r
diabetes_predicted = rep("0", 154)
diabetes_predicted[fitted.results > 0.5] = "1"
tab = table (diabetes_predicted, diabetes_testing$class)
tab
```

```
##
## diabetes_predicted  0  1
##                  0 89 24
##                  1 10 31
```

```r
precision = 89 / (89 + 24)
recall = 89 / (89 + 10)
accuracy = (89 + 21) / (89 + 24 + 10 + 31)
f1 = ((2*89)/(2*89+10+31))
precision
```

```
## [1] 0.7876106
```

```
recall
```

```
## [1] 0.8989899
```

```
accuracy
```

```
## [1] 0.7142857
```

```
f1
```

```
## [1] 0.8127854
```

after removing all the data that could make us do wrong predictions and after making our confusion table I have finly calculated the precision recall accuracy f1, the score are very high, therefore my trained model can survive.

3. Run K-NN on the same data (also using the same setup) to predict diabetes. Try different K's (K=1, K=3). Report the same scores as before (for each K value). Compare the models with F score. Which model has better Accuracy and F score? (logistic or KNN K1 or KNN K3). Optional: plot also roc curves to compare.

```
gp = runif(nrow(diabetes))
diabetes = diabetes[order(gp),]
diabetes$class = as.factor(diabetes$class)
str(diabetes)
```

```
## 'data.frame':    768 obs. of  9 variables:
##  $ times_pregnant: int  0 5 5 1 3 12 4 1 4 7 ...
##  $ glucose_to    : int  108 166 77 107 78 92 189 153 154 105 ...
##  $ diastolic_pg  : int  68 76 82 68 50 62 110 82 72 0 ...
##  $ triceps       : int  20 0 41 19 32 7 31 42 29 0 ...
##  $ insulin       : int  0 0 42 0 88 258 0 485 126 0 ...
##  $ mass_index    : num  27.3 45.7 35.8 26.5 31 27.6 28.5 40.6 31.3 0 ...
##  $ pedigree      : num  0.787 0.34 0.156 0.165 0.248 0.926 0.68 0.687 0.338 0.305 ...
##  $ age           : int  32 27 35 24 26 44 37 23 37 24 ...
##  $ class         : Factor w/ 2 levels "0","1": 1 2 1 1 2 2 1 1 1 1 ...
```

```
diabetes_n <- as.data.frame(lapply(diabetes[1:8], normalize))
diabetes_train = diabetes_n[1:614,]
diabetes_test = diabetes_n[615:768,]

diabetes_train_target = diabetes [1:614,9]
diabetes_test_target = diabetes[615:768, 9]
```

- k = 3

```r
modelc = knn(train = diabetes_train ,test = diabetes_test,cl = diabetes_train_target,k = 3 )
table(modelc)
```

```
## modelc
##   0   1
## 103  51
```

- k = 1

```r
modelse = knn(train = diabetes_train ,test = diabetes_test,cl = diabetes_train_target,k = 1)
table(modelse)
```

```
## modelse
##   0   1
## 100  54
```

k=1 Correlation coefficient 0.4132 Mean absolute error 0.2532 Root mean squared error 0.5032 Relative absolute error 56.5431 % Root relative squared error 107.6767 %

k=3 Correlation coefficient 0.4115 Mean absolute error 0.3009 Root mean squared error 0.4454 Relative absolute error 67.1751 % Root relative squared error 95.3091 %

4. In this task we are using diamonds data from the package ggplot2 (data(diamonds)). Build regression models predicting price from the rest of the features, where

A) model 1 has all the features
B) model 2 has all the features + 'carat' and 'depth' of degree 2
C) model 3 has all the features + 3rd degree polynomials of 'carat' and 'depth' (i.e. carat^3, carat^2, carat, depth^3,...)
D) model 4 has all the features + 3rd degree polynomials of 'carat' and 'depth' + 'x','y','z' of degree 2 in R you can use poly(x,d) to evaluate a polynomial of degree d, e.g. lm(price ~ poly(x,3) + ..., data=diamonds) Use the regular 80% train / 20% test split. Measure the RMSE for all the models on the train and test set and plot a graph, where on x-axis models are sorted according to the complexity of the model and on y-axis RMSE for train and test split. What do you observe? Can we diagnose under- or overfitting problems?

```
##
## Call:
## lm(formula = price ~ ., data = trained_diamond)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -21305.8   -591.3   -180.9    377.0  10697.3
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2167.521    453.064   4.784 1.72e-06 ***
## carat        11220.089     53.602 209.321  < 2e-16 ***
## cutGood        585.124     37.668  15.534  < 2e-16 ***
## cutIdeal       830.038     37.555  22.102  < 2e-16 ***
## cutPremium     761.013     36.148  21.053  < 2e-16 ***
## cutVery Good   720.612     36.273  19.866  < 2e-16 ***
```

```
## colorE        -212.910     19.926 -10.685  < 2e-16 ***
## colorF        -285.599     20.144 -14.178  < 2e-16 ***
## colorG        -491.916     19.736 -24.925  < 2e-16 ***
## colorH        -991.080     20.962 -47.279  < 2e-16 ***
## colorI       -1463.934     23.521 -62.240  < 2e-16 ***
## colorJ       -2379.003     29.197 -81.482  < 2e-16 ***
## clarityIF     5322.542     56.739  93.807  < 2e-16 ***
## claritySI1    3647.276     48.405  75.349  < 2e-16 ***
## claritySI2    2693.676     48.632  55.389  < 2e-16 ***
## clarityVS1    4577.668     49.451  92.570  < 2e-16 ***
## clarityVS2    4261.135     48.653  87.582  < 2e-16 ***
## clarityVVS1   4993.742     52.302  95.479  < 2e-16 ***
## clarityVVS2   4932.459     50.926  96.854  < 2e-16 ***
## depth          -63.063      5.007 -12.595  < 2e-16 ***
## table          -27.884      3.241  -8.604  < 2e-16 ***
## x            -1072.932     46.668 -22.991  < 2e-16 ***
## y               83.806     38.311   2.188   0.0287 *
## z              -37.472     34.479  -1.087   0.2771
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1126 on 43128 degrees of freedom
## Multiple R-squared:  0.9203, Adjusted R-squared:  0.9203
## F-statistic: 2.166e+04 on 23 and 43128 DF,  p-value: < 2.2e-16
```

```
##
## Call:
## lm(formula = price ~ poly(carat, 2) + poly(depth, 2) + ., data = trained_diamond)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13878.8   -587.7   -183.0    384.8  10637.5
##
## Coefficients: (2 not defined because of singularities)
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       1.336e+04  3.330e+02  40.121  < 2e-16 ***
## poly(carat, 2)1   1.321e+06  9.759e+03 135.403  < 2e-16 ***
## poly(carat, 2)2  -5.701e+04  2.159e+03 -26.409  < 2e-16 ***
## poly(depth, 2)1  -3.262e+04  1.575e+03 -20.708  < 2e-16 ***
## poly(depth, 2)2  -4.620e+03  1.397e+03  -3.308 0.000941 ***
## carat                    NA         NA      NA       NA
## cutGood           4.950e+02  4.116e+01  12.025  < 2e-16 ***
## cutIdeal          7.333e+02  4.388e+01  16.712  < 2e-16 ***
## cutPremium        6.734e+02  4.238e+01  15.889  < 2e-16 ***
## cutVery Good      6.118e+02  4.194e+01  14.589  < 2e-16 ***
## colorE           -2.117e+02  1.976e+01 -10.709  < 2e-16 ***
## colorF           -2.965e+02  1.998e+01 -14.837  < 2e-16 ***
## colorG           -5.044e+02  1.958e+01 -25.759  < 2e-16 ***
## colorH           -1.005e+03  2.080e+01 -48.317  < 2e-16 ***
## colorI           -1.467e+03  2.333e+01 -62.868  < 2e-16 ***
## colorJ           -2.363e+03  2.897e+01 -81.571  < 2e-16 ***
## clarityIF         5.235e+03  5.646e+01  92.724  < 2e-16 ***
## claritySI1        3.556e+03  4.823e+01  73.741  < 2e-16 ***
## claritySI2        2.605e+03  4.843e+01  53.788  < 2e-16 ***
```

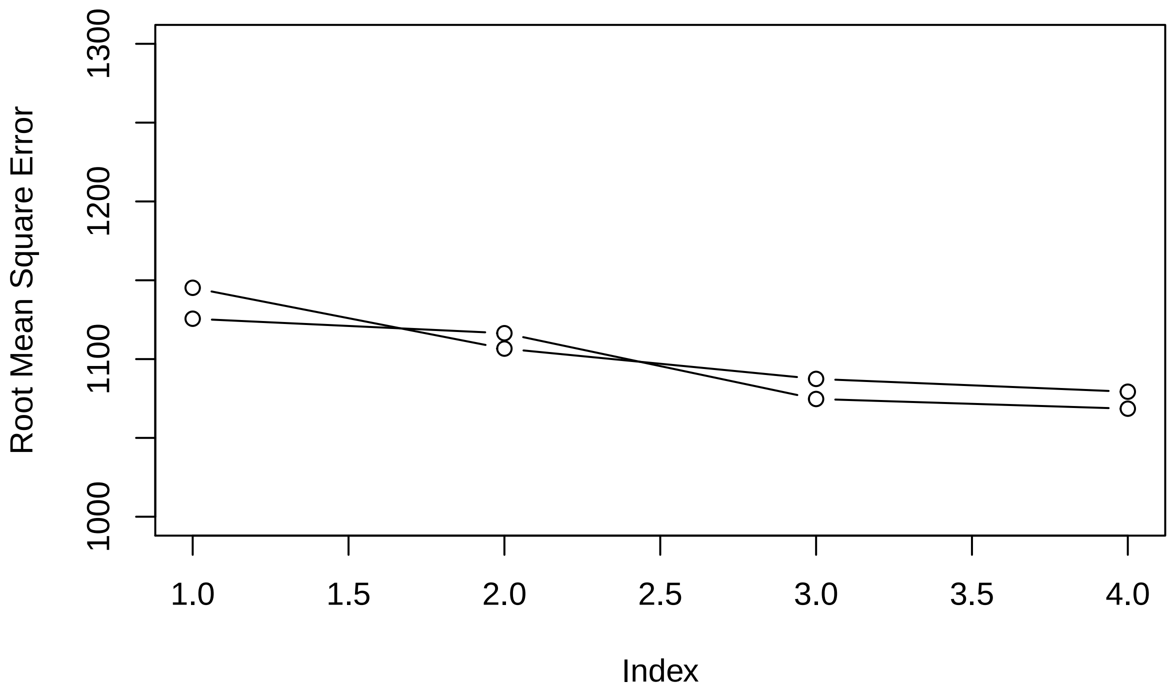
```
## clarityVS1       4.484e+03  4.927e+01  91.009  < 2e-16 ***
## clarityVS2       4.168e+03  4.848e+01  85.974  < 2e-16 ***
## clarityVVS1      4.903e+03  5.209e+01  94.136  < 2e-16 ***
## clarityVVS2      4.837e+03  5.074e+01  95.343  < 2e-16 ***
## depth                   NA         NA      NA       NA
## table           -3.623e+01  3.231e+00 -11.212  < 2e-16 ***
## x               -1.913e+03  5.631e+01 -33.972  < 2e-16 ***
## y               -1.915e+01  3.820e+01  -0.501 0.616110
## z               -5.580e+01  3.420e+01  -1.631 0.102835
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1117 on 43126 degrees of freedom
## Multiple R-squared:  0.9216, Adjusted R-squared:  0.9216
## F-statistic: 2.028e+04 on 25 and 43126 DF,  p-value: < 2.2e-16
```

```
##
## Call:
## lm(formula = price ~ poly(carat, 3) + poly(depth, 3) + ., data = trained_diamond)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11117.5   -549.4   -165.0    372.3  25146.7
##
## Coefficients: (2 not defined because of singularities)
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       4762.638    355.918  13.381  < 2e-16 ***
## poly(carat, 3)1 993849.677  10987.224  90.455  < 2e-16 ***
## poly(carat, 3)2   5704.077   2347.173   2.430 0.015095 *
## poly(carat, 3)3 -73910.443   1271.260 -58.140  < 2e-16 ***
## poly(depth, 3)1 -13833.504   1562.879  -8.851  < 2e-16 ***
## poly(depth, 3)2  -6654.064   1349.239  -4.932 8.18e-07 ***
## poly(depth, 3)3   4131.188   1100.974   3.752 0.000175 ***
## carat                   NA         NA      NA       NA
## cutGood            441.344     39.695  11.118  < 2e-16 ***
## cutIdeal           634.288     42.670  14.865  < 2e-16 ***
## cutPremium         550.561     41.212  13.359  < 2e-16 ***
## cutVery Good       541.503     40.649  13.321  < 2e-16 ***
## colorE            -216.788     19.027 -11.394  < 2e-16 ***
## colorF            -285.559     19.239 -14.843  < 2e-16 ***
## colorG            -503.814     18.850 -26.728  < 2e-16 ***
## colorH           -1020.824     20.023 -50.981  < 2e-16 ***
## colorI           -1504.947     22.469 -66.978  < 2e-16 ***
## colorJ           -2384.884     27.891 -85.509  < 2e-16 ***
## clarityIF         4995.159     54.517  91.626  < 2e-16 ***
## claritySI1        3377.011     46.559  72.532  < 2e-16 ***
## claritySI2        2426.026     46.752  51.891  < 2e-16 ***
## clarityVS1        4286.985     47.575  90.110  < 2e-16 ***
## clarityVS2        3972.618     46.808  84.870  < 2e-16 ***
## clarityVVS1       4675.625     50.309  92.939  < 2e-16 ***
## clarityVVS2       4623.788     48.999  94.366  < 2e-16 ***
## depth                   NA         NA      NA       NA
## table              -24.447      3.134  -7.801 6.30e-15 ***
## x                 -651.360     58.457 -11.142  < 2e-16 ***
```

```
## y                  126.527     36.877   3.431 0.000602 ***
## z                  -14.540     32.941  -0.441 0.658934
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1075 on 43124 degrees of freedom
## Multiple R-squared:  0.9274, Adjusted R-squared:  0.9273
## F-statistic: 2.039e+04 on 27 and 43124 DF,  p-value: < 2.2e-16
```

```
##
## Call:
## lm(formula = price ~ poly(carat, 3) + poly(depth, 3) + poly(x,
##     2) + poly(y, 2) + poly(z, 2) + ., data = trained_diamond)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -11298.6   -544.2   -162.7    365.8  22394.2
##
## Coefficients: (5 not defined because of singularities)
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        998.588    199.687   5.001 5.73e-07 ***
## poly(carat, 3)1 597402.247  20866.900  28.629  < 2e-16 ***
## poly(carat, 3)2  -7008.638   2524.303  -2.776 0.005498 **
## poly(carat, 3)3 -63463.514   1382.707 -45.898  < 2e-16 ***
## poly(depth, 3)1   7826.216   2090.636   3.743 0.000182 ***
## poly(depth, 3)2  -7059.297   1343.958  -5.253 1.51e-07 ***
## poly(depth, 3)3   2163.301   1100.330   1.966 0.049300 *
## poly(x, 2)1      49459.004  23344.066   2.119 0.034122 *
## poly(x, 2)2     111437.238   4998.150  22.296  < 2e-16 ***
## poly(y, 2)1     177594.072  19835.161   8.953  < 2e-16 ***
## poly(y, 2)2     -20607.274   2696.259  -7.643 2.17e-14 ***
## poly(z, 2)1      29957.344   9185.911   3.261 0.001110 **
## poly(z, 2)2      -6245.026   2084.263  -2.996 0.002735 **
## carat                   NA         NA      NA       NA
## cutGood            398.071     39.640  10.042  < 2e-16 ***
## cutIdeal           597.810     42.510  14.063  < 2e-16 ***
## cutPremium         502.688     41.040  12.249  < 2e-16 ***
## cutVery Good       501.662     40.622  12.350  < 2e-16 ***
## colorE            -218.050     18.919 -11.526  < 2e-16 ***
## colorF            -288.222     19.130 -15.066  < 2e-16 ***
## colorG            -506.215     18.743 -27.008  < 2e-16 ***
## colorH           -1028.313     19.915 -51.634  < 2e-16 ***
## colorI           -1514.458     22.346 -67.774  < 2e-16 ***
## colorJ           -2388.736     27.734 -86.130  < 2e-16 ***
## clarityIF         4952.972     54.295  91.223  < 2e-16 ***
## claritySI1        3341.669     46.360  72.081  < 2e-16 ***
## claritySI2        2391.314     46.545  51.377  < 2e-16 ***
## clarityVS1        4248.429     47.374  89.679  < 2e-16 ***
## clarityVS2        3938.312     46.604  84.506  < 2e-16 ***
## clarityVVS1       4630.456     50.104  92.416  < 2e-16 ***
## clarityVVS2       4577.851     48.806  93.797  < 2e-16 ***
## depth                   NA         NA      NA       NA
## table              -10.772      3.179  -3.388 0.000703 ***
## x                       NA         NA      NA       NA
```

```
## y                        NA         NA      NA       NA
## z                        NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1069 on 43121 degrees of freedom
## Multiple R-squared:  0.9282, Adjusted R-squared:  0.9282
## F-statistic: 1.858e+04 on 30 and 43121 DF,  p-value: < 2.2e-16
```



Overfitting occurs when a statistical model describes random error or noise instead of the underlying relationship. Overfitting generally occurs when a model is excessively complex, such as having too many parameters relative to the number of observations.

7. (optional bonus, 1p). Try ridge and lasso regressions for model 4 from task 4 and add the resulting RMSE of training and test set on the plot generated in task 4. Did it help?