Quiz3

Nam Jun Lee

11/04/2021

Q1 This problem is regarding writing your own code for K nearest neighbor classification.

##a. Import the training predictors (x), training response (y) and testing (x) data sets provided with this quiz. Merge the testing (x) and training (x) row-wise (using rbind()). The training (x) is a 100×4 data frame, the testing (x) is a 50×4 data frame, thus the combined data should be 150×4 where the firs 50 rows are of the testing data.

```
# merge testing (x) and training(x) row-wise
df <- rbind(testX, trainX)
# display df first 6 row
head(df)
```

```
##            V1        V2        V3         V4
## 1 1.145120601 2.2125916 1.8604150  0.8423812
## 2 2.573741105 1.4693364 1.2972234  1.1342930
## 3 1.028673799 2.5158459 0.2376316  2.1784457
## 4 1.170138729 1.4522630 1.7760037 -0.2764890
## 5 2.334068305 0.4221641 1.1313296 -0.3561858
## 6 0.009083285 0.4082762 0.6617695  0.5522097
```

b. Compute a matrix of distances amongst all observations of the combined data set of part (a). (use the function dist()).

```
# matrix of distances amongst combined data set
dst <- as.matrix(dist(df, method="euclidean"))
# display dst
head(dst)
```

```
##          1        2        3        4        5        6        7        8
## 1 0.000000 1.730833 2.126974 1.355626 2.566585 2.463145 2.281941 2.426449
## 2 1.730833 0.000000 2.386505 2.046931 1.844739 2.906190 2.153653 2.573264
## 3 2.126974 2.386505 0.000000 3.089420 3.648371 2.882005 2.969498 1.036757
## 4 1.355626 2.046931 3.089420 0.000000 1.684575 2.089550 2.223193 3.619948
## 5 2.566585 1.844739 3.648371 1.684575 0.000000 2.539964 2.280781 4.182943
## 6 2.463145 2.906190 2.882005 2.089550 2.539964 0.000000 1.712394 3.733768
##          9       10       11       12       13       14       15       16
```

```
## 1 2.2922620 2.627640 2.178753 3.290777 2.794907 1.881147 3.675566 2.345826
## 2 0.8894629 1.961059 1.394363 3.619482 2.688433 1.306571 4.610634 2.327660
## 3 2.9353992 2.433159 2.383991 4.263445 3.719397 1.768985 2.952590 1.553738
## 4 2.0840846 2.481171 2.177198 2.548598 1.708553 2.511139 4.006641 2.679460
## 5 1.1722028 1.973937 1.781700 2.753380 1.334293 2.588596 4.896067 2.881385
## 6 2.7309901 1.770392 1.895562 1.582449 2.203400 2.438862 2.625906 1.634386
##         17       18       19        20       21       22        23       24
## 1 1.483272 2.167165 2.821822 1.9811497 2.604973 2.697453 2.1933948 2.186064
## 2 2.693671 2.977092 2.274640 2.2355250 1.932530 2.451753 0.9272508 1.583028
## 3 2.385029 2.271194 2.143689 3.7084778 3.193763 3.079291 2.2264516 2.622869
## 4 1.561766 2.152941 3.522932 0.7382091 2.131965 2.850920 2.8247630 1.755087
## 5 2.821944 2.996571 3.508222 1.3463724 1.388713 2.840136 2.6704208 1.244956
## 6 1.369148 1.116360 3.013701 2.4980820 1.799241 2.189974 3.5959982 2.050161
##         25       26       27       28       29       30       31       32
## 1 1.876149 3.354727 2.492760 1.426461 4.643174 1.398835 3.891529 2.987553
## 2 3.112226 2.714225 2.311210 2.138408 3.875483 1.181838 3.271154 2.514261
## 3 2.261735 3.867254 1.371007 2.542518 5.547284 2.445141 4.806031 3.943820
## 4 2.452785 4.110499 3.519821 2.337195 5.125360 2.197396 4.459069 3.686712
## 5 3.704643 4.216585 3.843147 3.205882 4.975433 2.668305 4.453066 3.894900
## 6 3.119291 5.457530 3.289713 2.853993 6.458235 3.508914 5.565268 5.033843
##         33       34       35       36       37       38       39       40
## 1 4.394331 3.566810 2.102227 2.321231 1.972278 3.192783 2.970438 3.133886
## 2 3.632427 4.098578 2.384772 2.544909 1.982632 3.181072 3.161865 2.020205
## 3 4.566768 3.639933 2.228108 1.245968 2.421220 3.915223 3.254486 3.244143
## 4 5.261827 4.859587 3.373951 3.535255 2.995238 4.190580 3.824662 3.729229
## 5 5.230303 5.746002 4.072308 4.163891 3.495575 4.723014 4.340202 3.410798
## 6 6.125418 5.602482 4.054302 3.900335 3.465967 5.343449 3.760452 4.059687
##         41       42       43       44       45       46       47       48
## 1 1.401540 1.756784 2.674986 2.905392 3.120653 1.690466 2.852419 4.014796
## 2 2.032689 1.933008 3.254645 3.110567 3.620121 2.255608 2.420169 2.520016
## 3 2.328702 3.197821 2.711051 2.496208 3.844966 1.945187 3.432396 3.545467
## 4 2.529326 2.345774 3.590689 4.157867 4.243948 3.010614 3.528733 4.467609
## 5 3.369296 2.928951 4.458392 4.782363 5.116841 3.849776 3.769356 3.719859
## 6 3.189759 3.364098 4.681549 4.387954 5.442710 3.747410 4.991023 4.550758
##         49       50       51       52       53       54       55       56
## 1 3.107708 3.203587 2.637582 2.556212 2.623490 2.870005 3.134331 2.861687
## 2 2.666094 2.369405 2.931333 2.924125 2.053384 2.325459 3.049456 2.779079
## 3 3.327190 3.480656 3.909404 3.554717 3.284734 2.122523 3.437111 4.399052
## 4 4.038154 4.038371 1.485463 1.488005 2.630346 2.942321 2.620291 2.007600
## 5 4.238895 4.024823 1.832792 1.853296 2.355326 2.620659 2.566980 1.852102
## 6 4.533295 4.898658 2.660853 1.569717 2.453668 2.001779 3.199602 2.651688
##          57       58       59       60        61       62        63       64
## 1 1.4470252 2.359946 3.851538 1.401567 1.1663324 1.408707 0.9147213 3.028771
## 2 0.8007927 2.346070 3.030767 1.672870 1.9812316 2.342589 1.7995588 3.431112
## 3 1.9129466 3.838582 3.159103 2.082111 2.9548380 1.945105 2.4032188 3.621695
## 4 1.8094205 1.362160 3.800165 2.218416 0.8047617 2.344371 1.9592667 2.181237
## 5 1.9380056 1.309825 3.095502 2.808107 1.9995890 3.301640 2.9216887 2.484005
## 6 2.5941213 2.106101 2.595905 2.430875 1.9729955 2.281023 2.8866359 1.430731
##          65       66       67       68        69       70       71       72
## 1 0.5356456 2.608542 3.291190 2.409469 2.2941191 3.127014 3.075390 2.484990
## 2 1.7357617 2.440682 3.178609 2.772736 2.5030031 3.978021 2.751285 2.384549
## 3 2.3368244 4.006758 3.511881 2.818198 3.0135170 2.893994 4.426821 3.138233
## 4 0.8852928 1.418773 2.562341 2.416704 1.6146218 3.444706 2.168559 1.697179
## 5 2.1877774 1.025682 2.226005 2.851202 1.8271469 4.270409 1.591059 1.455529
```

```
## 6 2.2182952 2.785463 1.566893 1.125988 0.7417557 1.906586 2.604666 1.244792
##         73       74       75       76       77       78       79       80
## 1 3.061190 2.508638 3.815767 1.896446 3.416466 1.900865 3.605274 3.046881
## 2 3.859320 2.996392 2.736775 2.342622 2.927220 2.354570 3.419461 1.817883
## 3 3.860803 3.344532 3.073759 2.268426 5.027584 3.527863 2.350869 3.112717
## 4 3.144956 2.821340 3.730576 2.537409 3.204963 1.160437 3.760161 2.749319
## 5 3.952161 3.441030 2.841948 3.165940 3.014333 1.869034 3.722848 1.603405
## 6 2.331362 2.458845 3.229742 2.118945 4.562430 1.936009 2.969730 2.603233
##         81       82        83       84       85       86        87       88
## 1 2.887950 1.587398 1.4236800 2.603696 1.402205 2.353662 2.0345447 2.591028
## 2 2.818526 1.943492 2.6046342 1.711038 1.641234 2.616824 2.4779845 3.086295
## 3 2.647361 1.378807 3.4614795 3.521932 2.089463 1.944062 2.8787973 2.657232
## 4 2.790506 1.970500 0.9187288 2.584870 1.773973 2.381163 1.6999579 2.707865
## 5 2.762146 2.530465 2.4942406 2.282789 2.284861 2.740092 2.2410748 3.240936
## 6 1.097205 1.696711 2.6256537 4.117844 1.653413 1.364442 0.8027209 1.098295
##          89       90       91       92       93       94       95       96
## 1 1.9640583 2.039755 2.928607 2.640792 2.379452 2.701523 3.509442 4.052405
## 2 2.1687403 1.436646 2.737614 1.829360 3.620918 2.692772 3.527677 5.419265
## 3 3.4018582 2.049455 3.234469 2.599492 3.428381 2.071470 3.610119 5.155290
## 4 0.7106045 2.112384 2.241382 2.426098 1.990828 2.694787 3.756281 3.708963
## 5 1.1694815 1.899000 1.859304 1.752969 3.291582 2.785246 3.910441 5.000020
## 6 2.0049451 1.764433 1.350025 1.904549 1.663804 1.623144 2.660169 3.298119
##         97       98       99      100      101      102      103      104
## 1 2.141781 1.956374 2.424221 2.802349 2.816489 2.608641 2.216702 2.602077
## 2 1.039818 3.480066 1.376567 2.818887 2.454401 2.926719 2.381782 1.848929
## 3 1.802260 3.273997 3.650251 1.843647 2.695539 3.245501 1.456203 3.059175
## 4 2.576806 2.225758 1.943389 2.986477 3.869827 3.752626 3.471507 3.441873
## 5 2.340541 3.718392 1.066276 3.137520 4.149280 4.506457 4.070093 3.535736
## 6 2.927696 2.422876 3.228199 1.788858 4.238976 4.910975 3.753051 4.397424
##        105      106      107      108      109      110      111      112
## 1 2.844138 2.517072 4.543147 3.108983 2.824147 1.878896 3.032560 3.326799
## 2 2.706081 1.823240 3.400628 2.954860 2.570146 1.675874 2.650445 2.973241
## 3 3.102578 3.309632 4.907248 2.923469 4.035565 3.319648 3.598767 3.352271
## 4 3.969241 3.241915 5.051763 4.308103 3.344996 2.227294 3.965959 4.402030
## 5 4.442739 3.358730 4.590167 4.758680 3.603213 2.623616 4.272386 4.735007
## 6 4.917388 4.419984 5.739748 4.987696 4.364916 3.954899 5.122058 5.398726
##        113      114      115      116      117      118      119      120
## 1 3.440283 1.537325 2.428781 1.592852 3.102948 3.493665 3.600922 2.334287
## 2 3.441249 2.186351 1.935726 2.701353 3.006699 3.660234 3.695589 1.775265
## 3 2.485161 2.386566 3.540113 3.210708 1.849114 3.815494 3.236064 2.490847
## 4 4.707739 2.559240 3.049824 2.524648 4.161324 4.667571 4.827603 3.265479
## 5 5.232086 3.391682 3.252458 3.720873 4.511015 5.327845 5.387494 3.508458
## 6 5.128013 2.982238 4.305481 3.852721 3.745565 5.657431 5.084769 4.304541
##        121      122      123      124      125      126      127      128
## 1 4.324586 2.243679 4.528573 3.728667 3.031215 1.869835 2.302035 1.099117
## 2 3.823354 3.188481 3.924748 3.970776 2.518975 2.225962 3.100759 2.124337
## 3 4.105328 3.369485 4.726592 3.488167 1.804422 2.768360 3.113984 1.641883
## 4 5.407843 3.322492 5.179149 4.983656 4.085294 2.976580 3.505381 2.336871
## 5 5.596037 4.450748 5.083864 5.695455 4.259634 3.713403 4.533329 3.377491
## 6 6.095809 4.627110 5.288061 5.868400 4.340878 3.917476 4.590628 3.194140
##        129      130      131      132      133      134      135      136
## 1 2.417305 3.052088 2.820471 3.007001 2.355224 4.124173 2.154494 2.897326
## 2 2.590498 2.452785 2.549316 3.090025 2.301228 3.731504 1.046293 3.541082
## 3 4.002435 3.042266 3.815664 3.380461 2.628736 4.319236 2.292568 3.668918
```

```
## 4 2.845892 4.069832 3.601056 4.095856 3.493975 4.945019 2.913326 3.898940
## 5 3.404386 4.243061 3.947079 4.710217 3.998961 5.078160 2.866406 4.782220
## 6 4.092928 4.840101 4.845328 5.321115 4.189919 5.180874 3.600022 4.423808
##        137      138      139      140      141      142      143      144
## 1 2.324116 1.965850 2.373227 1.800320 2.364357 5.051635 2.717067 2.363884
## 2 2.411051 1.604907 1.553853 2.452500 3.509482 4.650804 2.339193 3.259136
## 3 2.690916 2.090005 3.199527 2.543545 2.915966 5.197920 3.608039 2.633863
## 4 3.330852 3.016991 2.957166 3.042447 3.593253 5.959134 3.519752 3.360222
## 5 3.851655 3.360654 2.945417 3.941316 4.829054 6.112277 3.820357 4.336265
## 6 3.661039 3.603016 3.852359 3.902286 4.552650 6.390030 4.795975 3.198627
##        145      146      147      148      149       150
## 1 2.727842 2.869806 1.398305 4.740699 3.315390 1.7069765
## 2 2.533968 3.107744 2.030465 4.066940 3.334460 0.9226643
## 3 2.214275 2.250630 1.793976 4.323557 3.840425 2.4312603
## 4 3.933302 4.045611 2.716841 5.739484 4.362864 2.1408055
## 5 4.355430 4.645344 3.573326 5.741334 4.932964 2.2420634
## 6 4.460037 3.928557 3.479356 6.051147 5.620891 3.3800846
```

c. For each testing data observation find the K=15 nearest training observation. Recall that in the distance matrix of part (b), the first 50 observations (rows) are testing and the remaining are training. To find the K observations with the least distances, use the following custom function knearest(). This will require you to setup a 50 $\times$ K matrix (say near.neigh), where each row represents each testing observation and each column has the corresponding K nearest neigbors. This will require a for loop over the number of testing observations.

```
# split test and train
# first 50 observaions are testing
test <- dst[1:50,]
# remaining are training
train <- dst[51:150,]
# function knearest (find smallest among all values of x index)
knearest=function(x,k=15){
n=length(x)
ind=c(1:n)
temp=cbind(ind,x)
temp.order=temp[order(x),]
knearest=temp.order[1:k,1]
return(knearest)
}
# find each testing observation K nearest neigbors (15)
a1 = knearest(test[1,])
a2 = knearest(test[2,])
a3 = knearest(test[3,])
a4 = knearest(test[4,])
a5 = knearest(test[5,])
a6 = knearest(test[6,])
a7 = knearest(test[7,])
a8 = knearest(test[8,])
a9 = knearest(test[9,])
a10 = knearest(test[10,])
```

```r
a11 = knearest(test[11,])
a12 = knearest(test[12,])
a13 = knearest(test[13,])
a14 = knearest(test[14,])
a15 = knearest(test[15,])
a16 = knearest(test[16,])
a17 = knearest(test[17,])
a18 = knearest(test[18,])
a19 = knearest(test[19,])
a20 = knearest(test[20,])
a21 = knearest(test[21,])
a22 = knearest(test[22,])
a23 = knearest(test[23,])
a24 = knearest(test[24,])
a25 = knearest(test[25,])
a26 = knearest(test[26,])
a27 = knearest(test[27,])
a28 = knearest(test[28,])
a29 = knearest(test[29,])
a30 = knearest(test[30,])
a31 = knearest(test[31,])
a32 = knearest(test[32,])
a33 = knearest(test[33,])
a34 = knearest(test[34,])
a35 = knearest(test[35,])
a36 = knearest(test[36,])
a37 = knearest(test[37,])
a38 = knearest(test[38,])
a39 = knearest(test[39,])
a40 = knearest(test[40,])
a41 = knearest(test[41,])
a42 = knearest(test[42,])
a43 = knearest(test[43,])
a44 = knearest(test[44,])
a45 = knearest(test[45,])
a46 = knearest(test[46,])
a47 = knearest(test[47,])
a48 = knearest(test[48,])
a49 = knearest(test[49,])
a50 = knearest(test[50,])
# merge all test observations in near.neigh
near.neigh = rbind(a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,
                   a11,a12,a13,a14,a15,a16,a17,a18,a19,a20,a21,a22,
                   a23,a24,a25,a26,a27,a28,a29,a30,a31,a32,a33,a34,a35,
                   a36,a37,a38,a39,a40,a41,a42,a43,a44,a45,a46,a47,a48,a49,a50)
# display near.neigh
near.neigh
```

```
##      1  65  63 128  61   4 147  30  41  60  85  62  83  28  57
## a1   1  65  63 128  61   4 147  30  41  60  85  62  83  28  57
## a2   2  57   9 150  23  97 135  30  14  99  11  90 139  24 138
## a3   3   8  36  27  82 103  16 128  14 147  97 125 100 117  57
## a4   4  89  20  61  65  83  78   1  58  66  51  52  17  69   5
```

```
## a5   5  66  99  89   9  24  58  13  20  21  72  71  80   4  92
## a6   6  69  87  81  88  18  68  72  91  86  17  64  67  52  12
## a7   7  22  53  68  85  87  21  11  76  74  60  90  78  88  14
## a8   8  36 103   3 125  27 145  46 147 113 117  35 138 128  44
## a9   9   2  99  80  11   5  24  57  90  21  92  97 150  10  23
## a10 10  92  90  54  11  80  24  21  16  81  94  91  72  59 100
## a11 11  90  92  10  21  14  85   9  80   7  24  54  53   2  16
## a12 12  87   6  69  68  78  58  71   7  56  72  88  73  21  52
## a13 13  52  51  66  89  72   5  91  24  64  67  69  20   4  55
## a14 14  60  85  11  19  90   2  27 135  76  97 138  57  37  16
## a15 15  70  18  86 100  88  17  94  16   6  93  82  81   3  68
## a16 16 100  54  86  94  82  90  81  10  11  92  85  18  14  88
## a17 17  18  93  87  65   6  82  98  61   1  85  69   4  62  86
## a18 18  86  17   6  82  94  69 100  93  16  87  88  64  81  72
## a19 19  14  27 117  60  76  22  40  37  11 138 146  85 137 101
## a20 20  89   4  66  58  78  61  83   5  51  65  13  52  99  56
## a21 21  11  92  90  10   7  80  72  69   9  53   5  24  58  87
## a22 22   7  53  95  68  76  74  85  11  60  14  87  21  19  88
## a23 23 135   2  97 150 120  57  30 138 104  14   9 106 139 125
## a24 24  92  90  72  10   9  57   5  80  11  91  13  21  89  69
## a25 25 128  65   1  17  82  43  18   3 147  57   4  86  83  36
## a26 26  47 111  32 112 143 106 120 149 132 104  50 105 130 131
## a27 27 117 103  19   8 146  14   3 138 125 101  36  44  37 145
## a28 28 114  41  63  60  62  42  37 137  76 126 140 144  85   1
## a29 29  31  32  33 131 107 143 111  50 115 106  26  38 109 104
## a30 30 150 110 120   2  23 135  57 147 106 138  63   1 128 115
## a31 31  29 131  32 109 115 143 107  50 106  33 111 139  38 104
## a32 32 143 131 106 111 115 104  38  26  50  31 139 105 149 109
## a33 33 121  50 130 111 148 107  31 112 104  49  38 105  32  29
## a34 34 118  45 124 127 102 108 119 105  44  35 140 149  38 141
## a35 35  46 133 140 147 126 145 103 138 105 108  44  37  41 102
## a36 36   8 103 125   3  46 147 145 113 128  35  27 138 120  44
## a37 37 137  41 138 114 133 126 101  28  60  35 140  49  63  42
## a38 38 111 131 118 105 149 102 143  45  32 133 126 112 132 108
## a39 39 137  37 144 114 136 146  28  76  49  74 101  41 134  60
## a40 40 139  49 135 101  50  48 104 138  37  19  14 106 137 107
## a41 41 114  28  63  37 126 140  60 137 147  46  42  62 138  35
## a42 42 129  63  28  41 109 114 126 139  37 115  60 137 116 140
## a43 43 128  36  25 141  47 147 132 120  46  30 122 150   8 124
## a44 44 146 119  35 101 103 133 145 108 137  37  46 117 140 138
## a45 45 102 118 127 149  38 122 132  34 105 140 111 126 141 124
## a46 46 147  35 140 103 126  41 133 128 138 127 145  36  37 102
## a47 47  26 120 110 132  30 150  84 106 112 143 111 149  32  23
## a48 48  40 135  23  19  50 107  14  97 139 104   2 138 101 130
## a49 49 101 133 137  37  50 134 130  40 104 138 139 108 126 105
## a50 50 104 130 106 111 143  49  33  32 105 131 139 115 133 101
```

d. Compute the posterior probability for each testing observation. For each testing observation, recover the training class corresponding to its K nearest neighbors and compute the mean.

```
# all training class corresponding to Knn and compute the mean
pr = c()
pr[1] = mean(train[near.neigh[1,]])
pr[2] = mean(train[near.neigh[2,]])
pr[3] = mean(train[near.neigh[3,]])
pr[4] = mean(train[near.neigh[4,]])
pr[5] = mean(train[near.neigh[5,]])
pr[6] = mean(train[near.neigh[6,]])
pr[7] = mean(train[near.neigh[7,]])
pr[8] = mean(train[near.neigh[8,]])
pr[9] = mean(train[near.neigh[9,]])
pr[10] = mean(train[near.neigh[10,]])
pr[11] = mean(train[near.neigh[11,]])
pr[12] = mean(train[near.neigh[12,]])
pr[13] = mean(train[near.neigh[13,]])
pr[14] = mean(train[near.neigh[14,]])
pr[15] = mean(train[near.neigh[15,]])
pr[16] = mean(train[near.neigh[16,]])
pr[17] = mean(train[near.neigh[17,]])
pr[18] = mean(train[near.neigh[18,]])
pr[19] = mean(train[near.neigh[19,]])
pr[20] = mean(train[near.neigh[20,]])
pr[21] = mean(train[near.neigh[21,]])
pr[22] = mean(train[near.neigh[22,]])
pr[23] = mean(train[near.neigh[23,]])
pr[24] = mean(train[near.neigh[24,]])
pr[25] = mean(train[near.neigh[25,]])
pr[26] = mean(train[near.neigh[26,]])
pr[27] = mean(train[near.neigh[27,]])
pr[28] = mean(train[near.neigh[28,]])
pr[29] = mean(train[near.neigh[29,]])
pr[30] = mean(train[near.neigh[30,]])
pr[31] = mean(train[near.neigh[31,]])
pr[32] = mean(train[near.neigh[32,]])
pr[33] = mean(train[near.neigh[33,]])
pr[34] = mean(train[near.neigh[34,]])
pr[35] = mean(train[near.neigh[35,]])
pr[36] = mean(train[near.neigh[36,]])
pr[37] = mean(train[near.neigh[37,]])
pr[38] = mean(train[near.neigh[38,]])
pr[39] = mean(train[near.neigh[39,]])
pr[40] = mean(train[near.neigh[40,]])
pr[41] = mean(train[near.neigh[41,]])
pr[42] = mean(train[near.neigh[42,]])
pr[43] = mean(train[near.neigh[43,]])
pr[44] = mean(train[near.neigh[44,]])
pr[45] = mean(train[near.neigh[45,]])
pr[46] = mean(train[near.neigh[46,]])
pr[47] = mean(train[near.neigh[47,]])
```

```r
pr[48] = mean(train[near.neigh[48,]])
pr[49] = mean(train[near.neigh[49,]])
pr[50] = mean(train[near.neigh[50,]])
# merge all posterior for testing observation in pr
pr = rbind(pr[1],pr[2],pr[3],pr[4],pr[5],pr[6],pr[7],pr[8],pr[9],pr[10],
           pr[11],pr[12],pr[13],pr[14],pr[15],pr[16],pr[17],pr[18],pr[19],
           pr[20],pr[21],pr[22],pr[23],pr[24],pr[25],pr[26],pr[27],pr[28],
           pr[29],pr[30],pr[31],pr[32],pr[33],pr[34],pr[35],pr[36],pr[37],
           pr[38],pr[39],pr[40],pr[41],pr[42],pr[43],pr[44],pr[45],pr[46],
           pr[47],pr[48],pr[49],pr[50])
# display pr
pr
```

```
##           [,1]
##  [1,] 2.662567
##  [2,] 2.666125
##  [3,] 2.560631
##  [4,] 2.671795
##  [5,] 2.969605
##  [6,] 2.595357
##  [7,] 2.248945
##  [8,] 2.579092
##  [9,] 2.848998
## [10,] 2.506007
## [11,] 2.568089
## [12,] 2.707201
## [13,] 2.509286
## [14,] 2.365167
## [15,] 2.509103
## [16,] 2.465648
## [17,] 2.953313
## [18,] 2.557034
## [19,] 2.630779
## [20,] 2.486294
## [21,] 2.665551
## [22,] 2.390446
## [23,] 2.868036
## [24,] 2.738793
## [25,] 2.733766
## [26,] 2.430746
## [27,] 2.916391
## [28,] 2.479589
## [29,] 2.392444
## [30,] 2.692789
## [31,] 2.410597
## [32,] 2.443445
## [33,] 2.391377
## [34,] 2.525182
## [35,] 2.502001
## [36,] 2.632386
## [37,] 2.464069
## [38,] 2.520722
## [39,] 2.667335
```

```
## [40,] 2.319404
## [41,] 2.494658
## [42,] 2.487184
## [43,] 2.693549
## [44,] 2.783705
## [45,] 2.505121
## [46,] 2.471228
## [47,] 2.558101
## [48,] 2.260587
## [49,] 2.410826
## [50,] 2.404711
```

e. Create a prediction variable pred with values 0, 1. Compare this prediction vector to the ytest variable to compute the total error in your classification.

```
# create prediction variable
pred <- rep(0, length(pr))
pred[pr > 2.5] <- 1
# display pred
pred
```

```
##  [1] 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 0 1 1 1 0 1 0 1 1 1 0 1 0 0 1 0 0 0 1 1 1 0 1
## [39] 1 0 0 0 1 1 1 0 1 0 0 0
```

```
# create ytest variable
ytest = c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
          0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
          1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1)
# display ytest table
table(ytest)
```

```
## ytest
##  0  1
## 50 50
```

```
# K = 3
# predictions data in the knn
pred.knn = knn(test, train, pred, k = 3)
# confusing matrix
table(pred.knn, ytest)
```

```
##         ytest
## pred.knn  0  1
##        0 11 24
##        1 39 26
```

```
# fraction of days for which the prediction was correct
mean(pred.knn == ytest)
```

```
## [1] 0.37
```

```r
# training error rate
mean(pred.knn != ytest)
```

```
## [1] 0.63
```

Percentage of current KNN [k = 3] predictions
Training correct prediction:
$\frac{(11+26)}{(11+24+39+26)} = 0.37$; **37.00 %**
Training error rate:
$1 - 0.37 = 0.63$; **63.00 %**

```r
# K = 7
# predictions data in the knn
pred.knn = knn(test, train, pred, k = 7)
# confusing matrix
table(pred.knn, ytest)
```

```
##          ytest
## pred.knn  0  1
##        0  5 27
##        1 45 23
```

```r
# fraction of days for which the prediction was correct
mean(pred.knn == ytest)
```

```
## [1] 0.28
```

```r
# training error rate
mean(pred.knn != ytest)
```

```
## [1] 0.72
```

Percentage of current KNN [k = 7] predictions
Training correct prediction:
$\frac{(4+19)}{(4+31+46+19)} = 0.23$; **23.00 %**
Training error rate:
$1 - 0.23 = 0.77$; **77.00 %**

```r
# K = 10
# predictions data in the knn
pred.knn = knn(test, train, pred, k = 10)
# confusing matrix
table(pred.knn, ytest)
```

```
##          ytest
## pred.knn  0  1
##        0  5 29
##        1 45 21
```

```r
# fraction of days for which the prediction was correct
mean(pred.knn == ytest)
```

```
## [1] 0.26
```

```r
# training error rate
mean(pred.knn != ytest)
```

```
## [1] 0.74
```

Percentage of current KNN [k = 10] predictions
Training correct prediction:
$\frac{(6+21)}{(6+29+44+21)} = 0.27$; **27.00 %**
Training error rate:
$1 - 0.27 = 0.73$; **73.00 %**

Since the test error rate is high, this classifier is judged to be not suitable for the model.