Shanghai Earthquake

Jiapeng Zhang NetID: jz704

Lemin Wu NetID: lw514

In this project, we are given earthquake data in area near Shanghai from 1977 to 2008. Different column means different data for different kinds of earthqaukes. Each magnitude is the average magitude within that two-week period.

We firstly try to train a linear model to predict average magnitude four weeks later after. If the predictd value is larger than 3.0 cut off, we think earthquake did *happended* during that time.

Linear Model

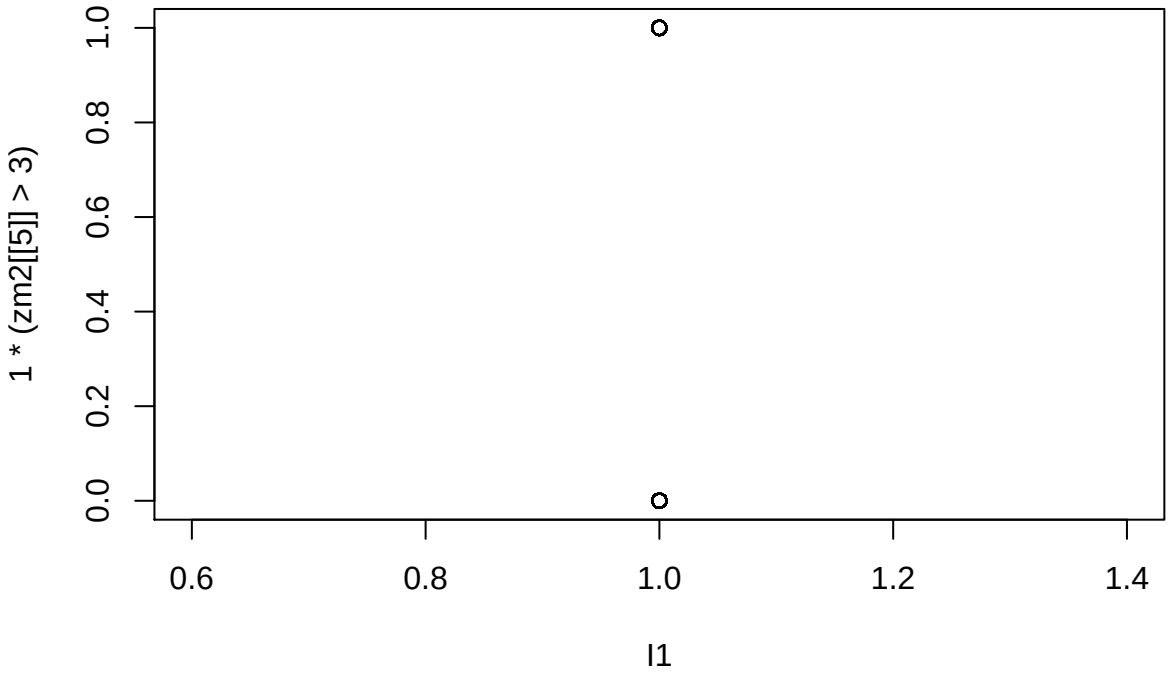
```
data<-read.csv("shanghai26.csv", header=TRUE)
data<-data[,-1]
rownum<-nrow(data)
x1<-data[[2]][1:(rownum-2)]
x2<-data[[2]][2:(rownum-1)]
x3<-data[[2]][3:rownum]
y1<-data[[3]][1:(rownum-2)]
y2<-data[[3]][2:(rownum-1)]
y3<-data[[3]][3:rownum]
zmat0<-cbind( x1, x2, y1, y2 , y3)
zmat0<-as.data.frame(zmat0)
zm1=zmat0[1:533,]
zm2=zmat0[401:(rownum-2), ]
fit_linear<-lm(y3~. , data=zm1 )
summary(fit_linear)
```

```
##
## Call:
## lm(formula = y3 ~ ., data = zm1)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.9561 -0.6852 -0.5895  1.2689  3.2574
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.307104   0.303206  10.907   <2e-16 ***
## x1           0.038283   0.040849   0.937    0.349
## x2           0.006823   0.040836   0.167    0.867
## y1           0.070358   0.043700   1.610    0.108
## y2          -0.021320   0.043617  -0.489    0.625
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.147 on 528 degrees of freedom
## Multiple R-squared:  0.007767,   Adjusted R-squared:  0.0002502
```

```
## F-statistic: 1.033 on 4 and 528 DF,  p-value: 0.3894
```

Based on the summary, we can see the fit is very bad. So the real magnitude values are very nonlinear. But we can still get some results if we close our eyes in linear model...

```
value_linear<-predict(fit_linear, newdata = zm2)
I1=1*(value_linear>3)
plot(I1, 1*(zm2[[5]]>3))
```




```
table(unname(I1), 1*(zm2[[5]]>3)  )
```

```
##
##       0   1
##   1 257 173
```

Ironically, the linear model gives all 1's. I.e., it says earthquakes happen at once every two weeks, not true!

Neural Networks

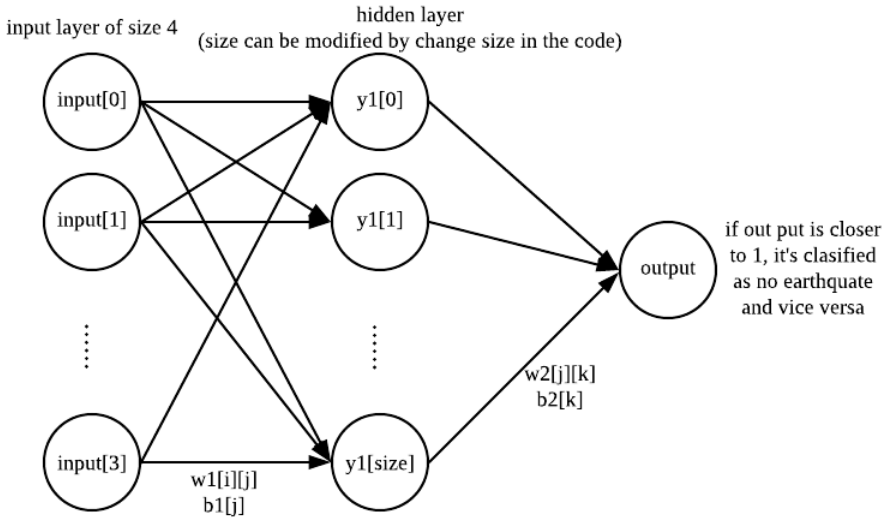
We are going to use nnet to nnet, which is a one-hidden-layner nerual net with logistic output. Loss function is L2 loss, backpropagation formulas are as follows:

$$\omega_{j,k}(t+1) = \omega_{j,k}(t) - \alpha \Delta_k \, out_j$$

where $\Delta_k = (out_k - y_k)g'(in_k)$

$$\omega_{i,j}(t+1) = \omega_{i,j}(t) - \alpha\Delta_j out_j$$

where $\omega_{i,j}(t+1) = \omega_{i,j}(t) - \alpha\Delta_j\, out_i$



For first magnitude:

```
library(nnet)
p3.data<-read.csv("shanghai26.csv", header = FALSE)[-1,-1]
p3.data<-p3.data[2:5]
p3.data<-apply(p3.data,1:2, as.numeric)

data.mag1<-p3.data[1:832,1]
v1<-data.mag1[1:600]
v2<-data.mag1[2:601]
v3<-data.mag1[3:602]
v4<-data.mag1[4:603]
train.input<-data.frame(cbind(v1,v2,v3,v4))
#names(train.input)<-c('X1','X2','X3','X4')

tar<-data.mag1[5:604]
tar<-tar>3
tar<-1*tar
train.target<-data.frame(tar)
#names(train.target)<-c('Y')

dat.in<-data.frame(train.target,train.input)

mod1<-nnet(train.input,train.target,data=dat.in,size=100, maxit = 2000,linout = F)
```

```
## # weights:  601
## initial  value 290.151609
## final  value 182.000000
## converged
```

```
val1<-data.mag1[605:828]
val2<-data.mag1[606:829]
val3<-data.mag1[607:830]
val4<-data.mag1[608:831]
val.input<-data.frame(cbind(val1,val2,val3,val4))
p<-predict(mod1,val.input)

p<-p>=0.5
p<-1*p
#sum(p)

val.target<-data.mag1[609:832]
val.target<-val.target>3
val.target<-data.frame(1*val.target)
#sum(val.target)

r1<-p*val.target
r2<-(1-p)*(1-val.target)
r3<-p*(1-val.target)
r4<-(1-p)*(val.target)
#sum(r1) #
#sum(r2)
#sum(r3)
#sum(r4)
table(p, data.mag1[609:832]>3)
```

```
##
## p   FALSE TRUE
##   0   107  117
```

```
sprintf("accuracy=%f", sum(p=(1* (data.mag1[609:832]) >3) ) / length(p) )
```

```
## [1] "accuracy=0.522321"
```

Then, the second magnitude:

```
#p3.data<-read.csv("shanghai26.csv", header = FALSE)[-1,-1]
#p3.data<-p3.data[2:5]
#p3.data<-apply(p3.data,1:2, as.numeric)

#data.mag1<-p3.data[1:832,1]
data.mag1<-data[[3]]
v1<-data.mag1[1:600]
v2<-data.mag1[2:601]
v3<-data.mag1[3:602]
v4<-data.mag1[4:603]
train.input<-data.frame(cbind(v1,v2,v3,v4))
#names(train.input)<-c('X1','X2','X3','X4')

tar<-data.mag1[5:604]
tar<-tar>3
tar<-1*tar
```

```r
train.target<-data.frame(tar)
#names(train.target)<-c('Y')

dat.in<-data.frame(train.target,train.input)

mod1<-nnet(train.input,train.target,data=dat.in,size=100, maxit = 2000,linout = F)
```

```
## # weights:  601
## initial  value 132.762590
## iter  10 value 115.810570
## iter  20 value 114.626901
## iter  30 value 112.961831
## iter  40 value 111.456369
## iter  50 value 109.498240
## iter  60 value 107.477502
## iter  70 value 105.044877
## iter  80 value 102.221055
## iter  90 value 99.627540
## iter 100 value 96.777720
## iter 110 value 94.810912
## iter 120 value 92.480717
## iter 130 value 90.515786
## iter 140 value 89.322391
## iter 150 value 88.255277
## iter 160 value 86.735834
## iter 170 value 85.103533
## iter 180 value 83.817437
## iter 190 value 82.766410
## iter 200 value 81.969944
## iter 210 value 81.143362
## iter 220 value 80.215020
## iter 230 value 79.923889
## iter 240 value 79.548538
## iter 250 value 78.983256
## iter 260 value 78.373426
## iter 270 value 77.393868
## iter 280 value 77.098286
## iter 290 value 76.996299
## iter 300 value 76.942154
## iter 310 value 76.909683
## iter 320 value 76.879089
## iter 330 value 76.855513
## iter 340 value 76.831321
## iter 350 value 76.812221
## iter 360 value 76.787134
## iter 370 value 76.773896
## iter 380 value 76.762538
## iter 390 value 76.753552
## iter 400 value 76.741779
## iter 410 value 76.718659
## iter 420 value 76.634360
## iter 430 value 76.544241
## iter 440 value 76.501366
```

```
## iter 450 value 76.451152
## iter 460 value 76.429383
## iter 470 value 76.419274
## iter 480 value 76.412741
## iter 490 value 76.409651
## iter 500 value 76.406848
## iter 510 value 76.399577
## iter 520 value 76.393375
## iter 530 value 76.379546
## iter 540 value 76.367711
## iter 550 value 76.353934
## iter 560 value 76.349027
## iter 570 value 76.346015
## iter 580 value 76.342135
## iter 590 value 76.336947
## iter 600 value 76.331225
## iter 610 value 76.327915
## iter 620 value 76.325131
## iter 630 value 76.320770
## iter 640 value 76.315897
## iter 650 value 76.312606
## iter 660 value 76.308088
## iter 670 value 76.298885
## iter 680 value 76.297915
## iter 690 value 76.294402
## iter 700 value 76.280336
## iter 710 value 76.251721
## iter 720 value 76.226504
## iter 730 value 76.197880
## iter 740 value 76.173532
## iter 750 value 76.152597
## iter 760 value 76.136317
## iter 770 value 76.120057
## iter 780 value 76.101567
## iter 790 value 76.081248
## iter 800 value 76.064280
## iter 810 value 76.053407
## iter 820 value 76.044858
## iter 830 value 76.030029
## iter 840 value 76.012849
## iter 850 value 75.991544
## iter 860 value 75.977402
## iter 870 value 75.960620
## iter 880 value 75.952426
## iter 890 value 75.937099
## iter 900 value 75.912849
## iter 910 value 75.886235
## iter 920 value 75.865440
## iter 930 value 75.849723
## iter 940 value 75.831565
## iter 950 value 75.810731
## iter 960 value 75.795382
## iter 970 value 75.779385
## iter 980 value 75.769501
```

```
## iter 990 value 75.763619
## iter1000 value 75.755931
## iter1010 value 75.743854
## iter1020 value 75.731918
## iter1030 value 75.720692
## iter1040 value 75.711194
## iter1050 value 75.705749
## iter1060 value 75.698660
## iter1070 value 75.694100
## iter1080 value 75.686299
## iter1090 value 75.681052
## iter1100 value 75.673945
## iter1110 value 75.668397
## iter1120 value 75.663973
## iter1130 value 75.661383
## iter1140 value 75.655726
## iter1150 value 75.653380
## iter1160 value 75.650387
## iter1170 value 75.649263
## iter1180 value 75.647911
## iter1190 value 75.646707
## iter1200 value 75.645448
## iter1210 value 75.644577
## iter1220 value 75.643122
## iter1230 value 75.641987
## iter1240 value 75.639229
## iter1250 value 75.638200
## iter1260 value 75.637144
## iter1270 value 75.636246
## iter1280 value 75.635310
## iter1290 value 75.633540
## iter1300 value 75.632526
## iter1310 value 75.630766
## iter1320 value 75.629364
## iter1330 value 75.627910
## iter1340 value 75.627464
## iter1350 value 75.626307
## iter1360 value 75.625074
## iter1370 value 75.623516
## iter1380 value 75.622264
## iter1390 value 75.620332
## iter1400 value 75.619358
## iter1410 value 75.618162
## iter1420 value 75.617593
## iter1430 value 75.616946
## iter1440 value 75.616180
## iter1450 value 75.614997
## iter1460 value 75.614318
## iter1470 value 75.612319
## iter1480 value 75.610379
## iter1490 value 75.609436
## iter1500 value 75.607935
## iter1510 value 75.607038
## iter1520 value 75.606417
```

```
## iter1530 value 75.606040
## iter1540 value 75.605777
## iter1550 value 75.605348
## iter1560 value 75.605118
## iter1570 value 75.604744
## iter1580 value 75.604353
## iter1590 value 75.603894
## iter1600 value 75.603609
## iter1610 value 75.603292
## iter1620 value 75.602925
## iter1630 value 75.602494
## iter1640 value 75.601994
## iter1650 value 75.601564
## iter1660 value 75.601106
## iter1670 value 75.600729
## iter1680 value 75.600549
## iter1690 value 75.600360
## iter1700 value 75.599815
## iter1710 value 75.599321
## iter1720 value 75.598866
## iter1730 value 75.598102
## iter1740 value 75.597232
## iter1750 value 75.596153
## iter1760 value 75.596067
## iter1770 value 75.595415
## iter1780 value 75.593068
## iter1790 value 75.591668
## iter1800 value 75.574779
## iter1810 value 75.511634
## iter1820 value 75.498614
## iter1830 value 75.496911
## iter1840 value 75.495773
## iter1850 value 75.494845
## iter1860 value 75.493544
## iter1870 value 75.492503
## iter1880 value 75.490820
## iter1890 value 75.489197
## iter1900 value 75.487152
## iter1910 value 75.485388
## iter1920 value 75.483344
## iter1930 value 75.481998
## iter1940 value 75.481666
## iter1950 value 75.481336
## iter1960 value 75.480998
## iter1970 value 75.480778
## iter1980 value 75.480630
## iter1990 value 75.480227
## iter2000 value 75.479885
## final  value 75.479885
## stopped after 2000 iterations
```

```
val1<-data.mag1[605:828]
val2<-data.mag1[606:829]
val3<-data.mag1[607:830]
```

```
val4<-data.mag1[608:831]
val.input<-data.frame(cbind(val1,val2,val3,val4))
p<-predict(mod1,val.input)

p<-p>=0.5
p<-1*p
#sum(p)

val.target<-data.mag1[609:832]
val.target<-val.target>3
val.target<-data.frame(1*val.target)
#sum(val.target)

r1<-p*val.target
r2<-(1-p)*(1-val.target)
r3<-p*(1-val.target)
r4<-(1-p)*(val.target)
#sum(r1) #
#sum(r2)
#sum(r3)
#sum(r4)
table(p, data.mag1[609:832]>3)
```

```
##
## p   FALSE TRUE
##   0    87   75
##   1    31   31
```

```
sprintf("accuracy=%f", sum(p=(1* (data.mag1[609:832]) >3) ) / length(p) )
```

```
## [1] "accuracy=0.473214"
```

The first is little better than random guessing, while the second is even slightly worse.

KNN

For first magnitude:

```
library(kknn)
library(class)
library(sampling)

p3.data<-read.csv("shanghai26.csv", header = FALSE)[-1,-1] #choose shanghai26.csv

p3.data<-p3.data[2:5]
p3.data<-apply(p3.data,1:2, as.numeric)

data.mag1<-p3.data[1:832,1]
v1<-data.mag1[1:600]
v2<-data.mag1[2:601]
```

```
v3<-data.mag1[3:602]
v4<-data.mag1[4:603]
train.input<-cbind(v1,v2,v3,v4)
#names(train.input)<-c('X1','X2','X3','X4')

tar<-data.mag1[5:604]
tar<-tar>3
tar<-1*tar
train.target<-tar
#names(train.target)<-c('Y')

val1<-data.mag1[605:828]
val2<-data.mag1[606:829]
val3<-data.mag1[607:830]
val4<-data.mag1[608:831]
val.input<-cbind(val1,val2,val3,val4)

mod2<-knn(train.input,val.input,train.target,k=15,prob=FALSE,use.all=TRUE)
mod2
```

```
##   [1] 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
##  [36] 0 0 0 0 0 0 1 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
##  [71] 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
## [106] 0 0 0 0 0 0 0 0 1 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
## [141] 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 1 0 0 0 0
## [176] 0 0 0 0 0 0 0 0 1 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
## [211] 1 0 0 0 0 0 0 0 0 1 0 0 0 0
## Levels: 0 1
```

```
p<-data.frame(mod2)
#sum(p)

val.target<-data.mag1[609:832]
val.target<-val.target>3
val.target<-data.frame(1*val.target)
#sum(val.target)

p<-apply(p,2, as.numeric)
r1<-p*val.target
r2<-(1-p)*(1-val.target)
r3<-p*(1-val.target)
r4<-(1-p)*(val.target)
#sum(r1)
#sum(r2)
#sum(r3)
#sum(r4)

table(p, data.mag1[609:832]>3)
```

```
##
## p   FALSE TRUE
##   0   103  112
##   1     4    5
```

```
sprintf("accuracy=%f", sum(p=(1* (data.mag1[609:832]) >3) ) / length(p) )
```

```
## [1] "accuracy=0.522321"
```

Then, for the second magnitude:

```
data.mag1<-p3.data[1:832,2]
v1<-data.mag1[1:600]
v2<-data.mag1[2:601]
v3<-data.mag1[3:602]
v4<-data.mag1[4:603]
train.input<-cbind(v1,v2,v3,v4)
#names(train.input)<-c('X1','X2','X3','X4')

tar<-data.mag1[5:604]
tar<-tar>3
tar<-1*tar
train.target<-tar
#names(train.target)<-c('Y')

val1<-data.mag1[605:828]
val2<-data.mag1[606:829]
val3<-data.mag1[607:830]
val4<-data.mag1[608:831]
val.input<-cbind(val1,val2,val3,val4)

mod2<-knn(train.input,val.input,train.target,k=15,prob=FALSE,use.all=TRUE)
mod2
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [36] 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
##  [71] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [106] 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
## [141] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [176] 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
## [211] 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## Levels: 0 1
```

```
p<-data.frame(mod2)
#sum(p)

val.target<-data.mag1[609:832]
val.target<-val.target>3
val.target<-data.frame(1*val.target)
#sum(val.target)

p<-apply(p,2, as.numeric)
r1<-p*val.target
r2<-(1-p)*(1-val.target)
r3<-p*(1-val.target)
r4<-(1-p)*(val.target)
#sum(r1)
```

```r
#sum(r2)
#sum(r3)
#sum(r4)

table(p, data.mag1[609:832]>3)
```

```
##
## p   FALSE TRUE
##   0   115  103
##   1     3    3
```

```r
sprintf("accuracy=%f", sum(p=(1* (data.mag1[609:832]) >3) ) / length(p) )
```

```
## [1] "accuracy=0.473214"
```

Still, one slightly better than random guess, the other being slighty worse.

Conclusion

So far we tried linear models, single layer neural networks and knn models. However, none of them performs very well, not even one exceeding 60% of accuracy. On one hand, earthquakes are very hard to predict; on the other, since the data is nonlinear, maybe we could try some more other models, deep neural nets, random forests, for exmaple. There's still room for future extension.