Shanghai Earthquake

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.

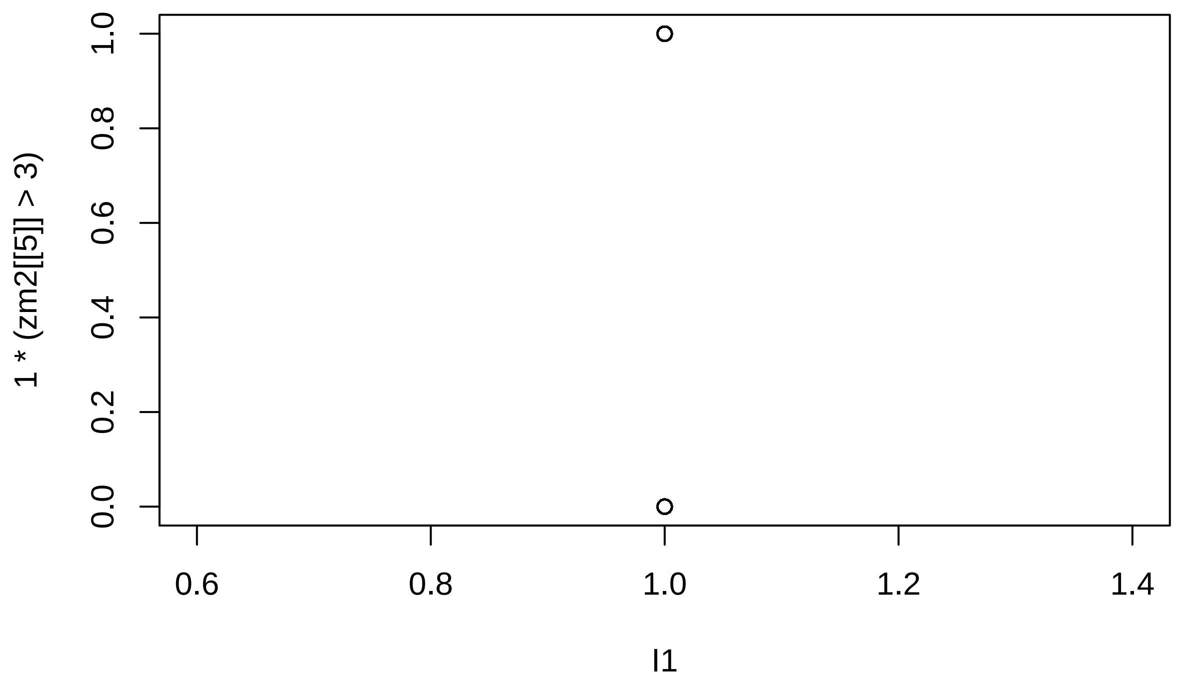
Linear Model

```
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

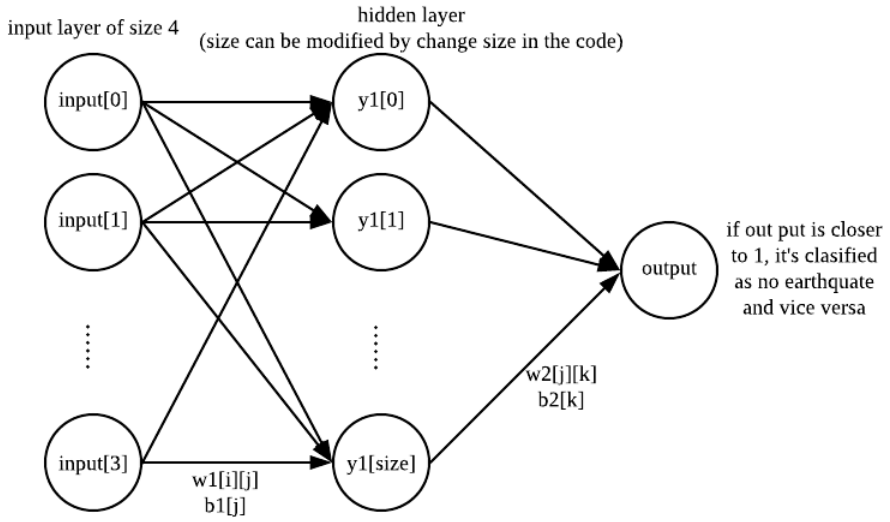
We are going to use nnet to nnet, which is a one-hidden-layner nerual net with logistic output. Loss function is L2 loss, backpropagation formulas are as follows:

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

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


input layer of size 4
hidden layer
(size can be modified by change size in the code)
input[0]
input[1]
input[3]
y1[0]
y1[1]
y1[size]
output
if out put is closer to 1, it's clasified as no earthquate and vice versa
w1[i][j]
b1[j]
w2[j][k]
b2[k]

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 131.184386
## iter  10 value 125.606322
## iter  20 value 124.398090
## iter  30 value 122.451347
## iter  40 value 119.164357
## iter  50 value 116.669274
## iter  60 value 112.744572
## iter  70 value 108.331892
## iter  80 value 105.492660
## iter  90 value 102.768970
## iter 100 value 99.741118
## iter 110 value 96.822754
## iter 120 value 93.911887
## iter 130 value 92.269573
## iter 140 value 90.843420
## iter 150 value 89.667124
## iter 160 value 87.418258
## iter 170 value 86.578576
## iter 180 value 85.747397
## iter 190 value 85.152535
## iter 200 value 84.735497
## iter 210 value 84.405626
## iter 220 value 84.282508
## iter 230 value 84.106741
## iter 240 value 84.001975
## iter 250 value 83.824659
## iter 260 value 83.659064
## iter 270 value 83.337066
## iter 280 value 83.047734
## iter 290 value 82.611246
## iter 300 value 82.411991
## iter 310 value 82.316951
## iter 320 value 82.237250
## iter 330 value 82.190005
## iter 340 value 82.158729
## iter 350 value 82.138627
## iter 360 value 82.114465
## iter 370 value 82.065373
## iter 380 value 82.017752
## iter 390 value 81.936471
## iter 400 value 81.885511
## iter 410 value 81.839008
## iter 420 value 81.811513
## iter 430 value 81.792131
## iter 440 value 81.779487
## iter 450 value 81.755090
## iter 460 value 81.706863
## iter 470 value 81.674036
## iter 480 value 81.631399
## iter 490 value 81.592069
## iter 500 value 81.469068
## iter 510 value 81.405140
## iter 520 value 81.327085
## iter 530 value 81.246797
## iter 540 value 81.168944
## iter 550 value 81.122304
## iter 560 value 81.042280
## iter 570 value 80.990715
## iter 580 value 80.976689
## iter 590 value 80.969178
## iter 600 value 80.959712
## iter 610 value 80.941849
## iter 620 value 80.929703
## iter 630 value 80.908506
## iter 640 value 80.879650
## iter 650 value 80.859226
## iter 660 value 80.832118
## iter 670 value 80.767430
## iter 680 value 80.660585
## iter 690 value 80.632164
## iter 700 value 80.600789
## iter 710 value 80.571473
## iter 720 value 80.545985
## iter 730 value 80.518515
## iter 740 value 80.492170
## iter 750 value 80.470349
## iter 760 value 80.451597
## iter 770 value 80.432567
## iter 780 value 80.421799
## iter 790 value 80.412832
## iter 800 value 80.403649
## iter 810 value 80.391812
## iter 820 value 80.385358
## iter 830 value 80.383126
## iter 840 value 80.379168
## iter 850 value 80.376785
## iter 860 value 80.375738
## iter 870 value 80.374507
## iter 880 value 80.371440
## iter 890 value 80.369133
## iter 900 value 80.366365
## iter 910 value 80.365316
## iter 920 value 80.364524
## iter 930 value 80.362518
## iter 940 value 80.361852
## iter 950 value 80.360688
## iter 960 value 80.359300
## iter 970 value 80.358646
## iter 980 value 80.357165
## iter 990 value 80.356339
## iter1000 value 80.355321
## iter1010 value 80.353227
## iter1020 value 80.352210
## iter1030 value 80.350843
## iter1040 value 80.349634
## iter1050 value 80.348619
## iter1060 value 80.348276
## iter1070 value 80.347159
## iter1080 value 80.345409
## iter1090 value 80.344309
## iter1100 value 80.343628
## iter1110 value 80.341860
## iter1120 value 80.340749
## iter1130 value 80.339371
## iter1140 value 80.338222
## iter1150 value 80.337450
## iter1160 value 80.336232
## iter1170 value 80.335009
## iter1180 value 80.333999
## iter1190 value 80.332840
## iter1200 value 80.330584
## final  value 80.330075
## 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    68   80
##   1    39   37
```

```
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]]
```

```r
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 353.516912
## final  value 164.000000
## converged
```

```r
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   118  106
```

```r
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:

```r
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 1 0 0 0 0 0 0
##  [36] 0 1 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 1 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 1 0 0 0 0 0 0 0 0 0 0 0 0 0
## [106] 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 0 0 0 0 0 0
## [141] 0 0 0 0 0 0 0 0 0 0 0 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 1 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
```

```r
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   100  111
##   1     7    6
```

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

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

Then, for the second magnitude:

```r
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 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 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 0
## [106] 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 0 0 0 0 1 0
## [141] 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 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 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)
#sum(r2)
#sum(r3)
#sum(r4)

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

```
##
## p   FALSE TRUE
##   0   115  104
##   1     3    2
```

```
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.