Rețele neuronale artificiale în R

Introducere

Obiectivul laboratorului este acela de a învăța despre rețelele neuronale artificiale (RNA) și implementarea lor în R. Tot codul folosit in aceasta prezentare este disponibil pe platforma moodle.

Rețelele neuronale sunt construite din unități numite perceptroni. Perceptronii au una sau mai multe intrări, o funcție de activare și o ieșire. O RNA este construita prin combinarea perceptronilor în mai multe straturi/nivele. Perceptronii dintr-un anumit nivel sunt independenți unul de celălalt, dar fiecare se conectează la toți perceptronii din următorul nivel.

Nivelul de intrare conține un perceptron pentru fiecare variabilă predictor (input). O RNA poate avea unul sau mai multe nivele ascunse care conțin un număr de perceptroni definit de utilizator. Fiecare perceptron din primul nivel ascuns primește o intrare de la fiecare perceptron din nivelul de intrare. Dacă există un al doilea nivel ascuns, fiecare perceptron din acest nivel primește o intrare de la fiecare perceptron din primul nivel ascuns, etc.

Nivelul de ieșire conține un perceptron pentru fiecare variabilă de răspuns (de regula una singura, dar uneori mai multe în situația unui răspuns multivariat). Fiecare perceptron de ieșire primește o intrare de la fiecare perceptron din nivelul ascuns final. Perceptronii au o funcție de activare neliniară (de exemplu, o funcție logistică) care determină valoarea lor de ieșire pe baza valorilor de intrare.

Conexiunile dintre perceptroni sunt caracterizate printr-o pondere. Mărimea ponderii controlează puterea influenței acelei intrări asupra perceptronului receptor. Semnul ponderii controlează dacă perceptronul din nivelul inferior stimulează sau inhibă semnalul către următorul nivel.

Ponderile sunt oarecum analoge coeficienților unui model de regresie liniar. Există, de asemenea, o ajustare aferenta deplasarii (bias) care reprezintă valoarea de bază a unui perceptron și este analogă cu termenul liber dintr-un model de regresie liniar. Dacă intrările sunt aproape de zero, deplasarea ne asigură că ieșirea retelei este aproape de valoarea medie. Cu toate acestea, situația este mult mai complexă datorită structurii rețelei. Acest lucru duce la relații complexe, neliniare între variabilele predictor și răspuns.

Pentru exemplificare vom folosi pachetul *neuralnet*, https://cran.r-project.org/web/packages/neuralnet/index.html

Un tutorial video despre modul de lucru cu RNA poate fi accesat aici: https://www.youtube.com/watch?v=lTMqXSSjCvk

Pachetul *neuralnet* este foarte flexibil, permitand utilizatorului sa construiasca RNA folosind o serie de algoritmi, functii de activare si functii de eroare, cu nivele ascunse multiple care permit modelare celor mai variate probleme.

Pentru exemplificare vom folosi setul de date Boston din pachetul MASS care contine o serie de variabile predictor pentru valoarea mediana a proprietatilor imobiliare din suburbiile orasului Boston. Obiectivul nostrum este acela de a utiliza toate variabilele (continue) disponibile in set pentru a prezice valoarea variabilei raspuns medv.

Vom face o comparatie intre rezultatele obtinute folosind un model de regresie linear si o retea neuronala artificiala.

```
set.seed(500)
library(MASS)
data <- Boston
```

Mai intai verificam ca nu exista date lipsa, altfel va trebuie sa imputam datele lipsa:

```
apply(data,2,function(x) sum(is.na(x)))
```

```
crim       zn    indus     chas      nox       rm      age      dis
rad      tax ptratio    black lstat     medv
       0        0        0        0        0        0        0        0
0        0        0        0   0        0
```

Nu exista data lipsa! Vom continua prin divizarea aleatoare a setului de date intr-un set de date de antrenare si un set de test, apoi vom calcula parametrii unui model de regresie liniara pentru setul de antrenare si ii vom testa folosind setul de date de test. Vom folosi functia `glm()` in loc de `lm()`. Motivele le vom vedea mai tarziu cand vom folosi tehnica validarii incrucisate.

```
index <- sample(1:nrow(data),round(0.75*nrow(data)))
```

```
train <- data[index,]
test <- data[-index,]
lm.fit <- glm(medv~., data=train)
summary(lm.fit)
pr.lm <- predict(lm.fit,test)
MSE.lm <- sum((pr.lm - test$medv)^2)/nrow(test)

Call:
glm(formula = medv ~ ., data = train)

Deviance Residuals:
     Min        1Q    Median        3Q       Max
-14.8563   -2.7227   -0.7165    1.8004   26.0196

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  38.352881   5.986496   6.407 4.58e-10 ***
crim         -0.070235   0.048155  -1.459 0.145555
zn            0.030454   0.016717   1.822 0.069305 .
indus        -0.001869   0.072787  -0.026 0.979531
chas          2.388145   1.020664   2.340 0.019831 *
nox         -17.721134   4.600002  -3.852 0.000138 ***
rm            3.673042   0.480320   7.647 1.83e-13 ***
age           0.004707   0.016005   0.294 0.768835
dis          -1.393532   0.245483  -5.677 2.80e-08 ***
rad           0.295930   0.083091   3.562 0.000417 ***
tax          -0.011487   0.004666  -2.462 0.014275 *
ptratio      -1.058387   0.159947  -6.617 1.30e-10 ***
black         0.010540   0.003245   3.248 0.001270 **
lstat        -0.525092   0.058972  -8.904  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

(Dispersion parameter for gaussian family taken to be 24.38024)

    Null deviance: 31783.5  on 379  degrees of freedom
Residual deviance:  8923.2  on 366  degrees of freedom
AIC: 2307.8

Number of Fisher Scoring iterations: 2
```

Functia `sample(x,size)` genereaza un vector cu dimensiunea specificata alcatuit din elemente selectate aleator dintre elemente vectorului `x`. Extragerea elementelor este fara inlocuire: `index` un vector aleator de indici. Deoarece luram cu un model de regresie liniara, vom

apela la Mean Squared Error (MSE – medie patratica a erorii) ca masura a distantei dintre predictii si datele reale.

Pregatirea datelor pentru utilizarea lor cu o RNA

Inainte de a folosi o retea neuronala, este nevoie de anumite prelucrari ale datelor.

Pasul 1. Prepocesarea datelor

O practica buna care este utilizata des este aceea de a normaliza datele inainte de utilizarea retelei. Normalizarea datelor este un pas foarte important: in functie de setul de date utilizat, fara normalizare se pot obtine rezultate fara valoare predictive sau se poate ingreuna foarte mult procesul de antrenare al retelei (algoritmul nu va converge). Exista diferite metode de normalizare a datelor: scalare min-max, z-normalization etc. In exemplul urmator vom folosi metoda min-max si vom scala datele in intervalul [0,1]. Scalarea datelor in intervalul [0,1] sau chiar [-1,1] conduce la rezultate bune in practica.

Vom scala si diviza setul de date:

```
maxs <- apply(data, 2, max)
mins <- apply(data, 2, min)

scaled <- as.data.frame(scale(data, center = mins, scale = maxs -
mins))

train_ <- scaled[index,]
test_ <- scaled[-index,]
```

Functia `scale()` returneaza o matrice care este apoi convertita la un `data.frame`.

Pasul 2. Parametrii

Nu exista nici o regula teoretica care sa ne spuna cate nivele si cati neuroni sa aiba o retea, dar sunt mai multe reguli practice. Din punct de vedere practic, un singur nivel ascuns este sufficient pentru foarte multe aplicatii. Numarul de neuroni din nivelul ascuns ar trebui sa fie cuprins intre

numarul de neuroni din nivelul de intrare sic el de iesire (se poate folosi ca valoare 2/3 din numarul neuronilor de intrare). Subliniem faptul ca acestea sunt doar reguli practice dar care nu garanteaza o solutie optima iar cea mai buna solutie este experimentarea.

In exemplul nostrum vom folosi o retea cu doua nivele ascunse in configuratia: 13:5:3:1. 13 reprezinta numarul variabilelor de intrare iar 1 reprezinta singura variabila de iesire (variabila raspuns `medv`).

```
n <- names(train_)
f <- as.formula(paste("medv ~", paste(n[!n %in% "medv"], collapse =
" + ")))
nn <- neuralnet(f,data=train_,hidden=c(5,3),linear.output=T)
```

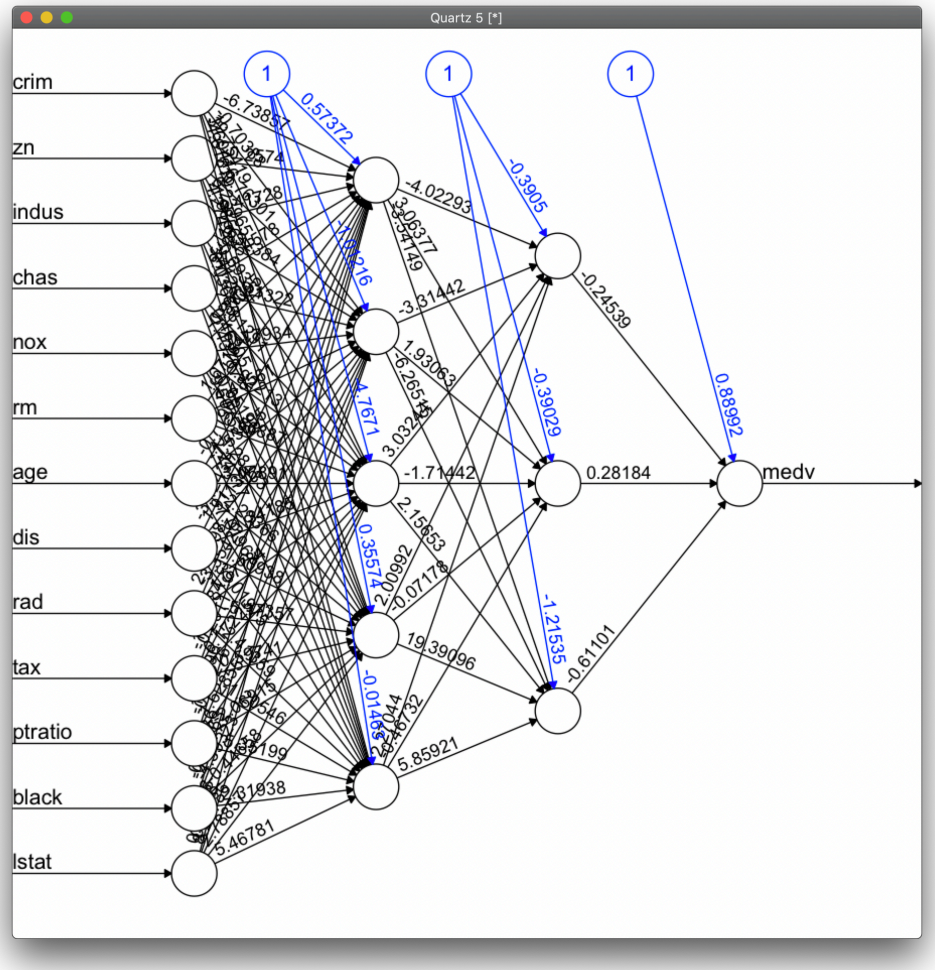
Observatii:

- Functia `neuralnet()` nu accepta formule cu `y~`. Main intai trebuie scrisa formula si apoi aceasta este transmisa ca parametru al functiei.
- Argumentul `hidden` accepta un vector ce specifica numarul de neuroni din fiecare nivel ascuns.
- Argumentul `linear.output=TRUE` specifica faptul ca dorim sa modelam o regresie, in timp ca valoarea `FALSE` specifica faptul ca vrem sa rezolvam o problema de clasificare.

Pachetul `neuralnet` furnizeaza o metoda foarte utila de reprezentare grafica a reteei:

```
plot(nn)
```

Mai jos este rezultatul functiei.



Liniile negre reprezinta conexiunile dintre fiecare nivel si ponderile asociate fiecarei conexiuni iar liniile albastre arata termenul de deplasare adaugat la fiecare pas. Deplasarea (bias) poate fi considerata similara cu termenul liber din modelul de regresie liniara.

Reteleaua neuronala este similara unei “cutii negre” si nu se pot spune multe lucruri despre procesul de calcul al parametrilor. Ceea ce ne intereseaza este faptul ca procesul de invatare a fost convergent iar reteaua este gata pentru a fi utilizata la predictii.

Pasul 3. Predictia valorilor medv folosind reteaua neuronala

Vom incerca sa efectuam predictii asupra valorilor variabilei raspuns medv din setul de test si vom calcula MSE. Trebuie sa tinem cont de faptul ca reteaua va produce rezultate normalizate si va trebui sa scalam aceste rezultate in sens invers pentru a obtine valori cu semnificatie.

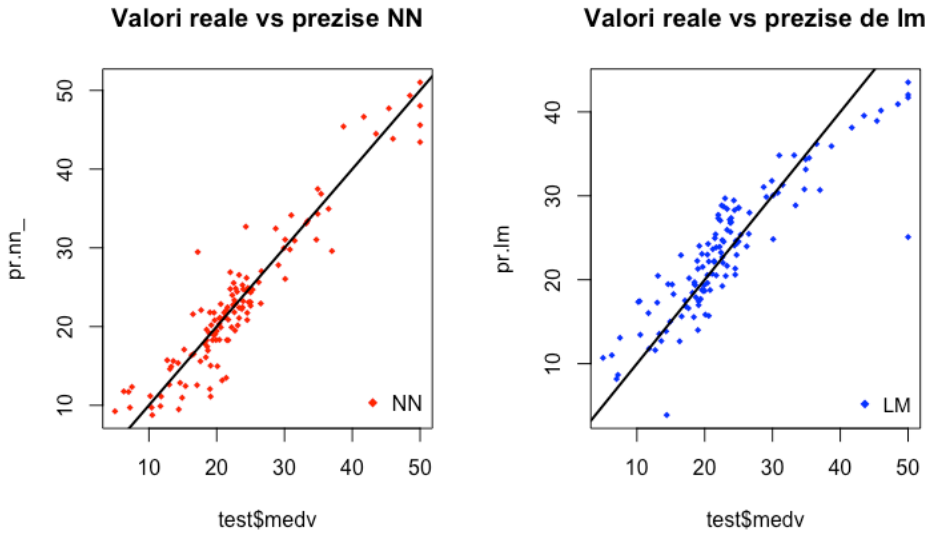
```
pr.nn <- neuralnet::compute(nn,test_[,1:13])

pr.nn_ <- pr.nn$net.result * (max(data$medv) - min(data$medv))+
min(data$medv)
test.r <- (test_$medv)*(max(data$medv) - min(data$medv))+
min(data$medv)

MSE.nn <- sum((test.r - pr.nn_)^2)/nrow(test_)
```

Vom compara acum valorile MSE pentru modelul de regresie liniara si al retelei neuronale:

```
> print(paste(MSE.lm,MSE.nn))
[1] "17.8602403063511 9.81345079680467"
```

Analizand rezultatele constatam ca reteaua neuronala a obtinut rezultate mai bune decat modelul de regresie liniara. Trebuie sa tinem insa cont de faptul ca aceste rezultate depind de modul de divizare a setului initial in seturi de antrenament si test.

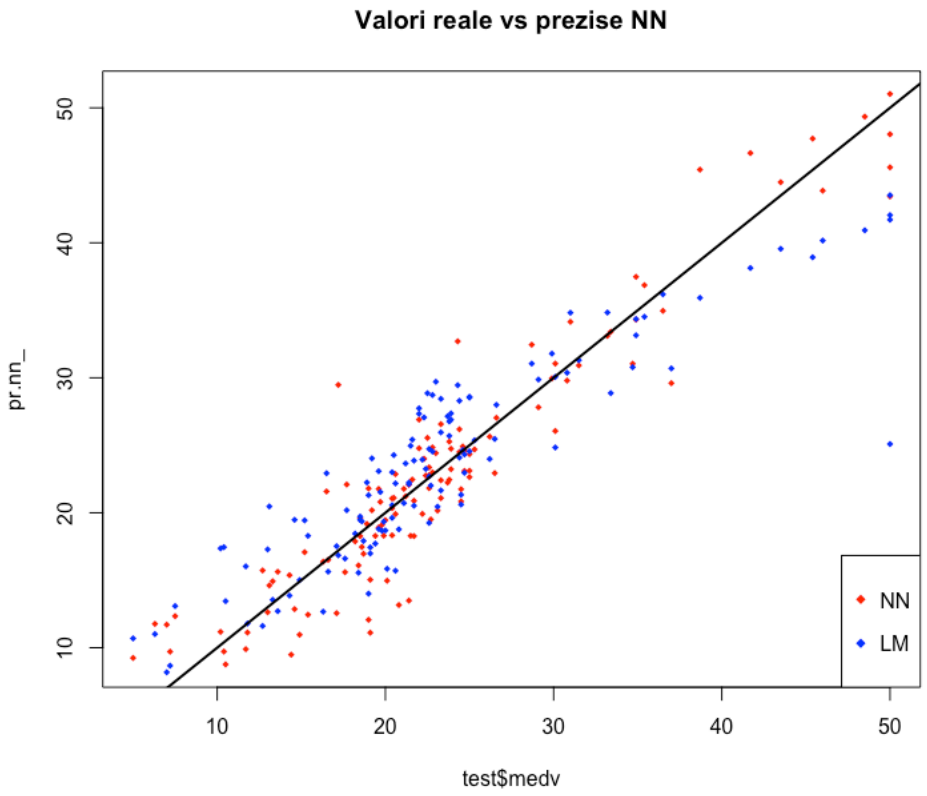
Vom aplica si metoda validarii incrucisate pentru a confirma rezultatele de mai sus. O comparatie intre performantele modelului de regresie liniara si ale retelei neuronale sunt prezentate mai jos.



O inspectie vizuala a acestor grafice ne arata faptul ca valorile prezise de reateaua neuronala sunt mai bine grupate in jurul valorilor reale (situatia ideala are fi cand toate valorile vor fi de-a lungul liniei diagonal, ceea ce este echivalent cu MSE = 0).

Un alt mod de inspectie vizuala este prezentat mai jos:

```
plot(test$medv,pr.nn_,col='red',main='Valori reale vs prezise
NN',pch=18,cex=0.7)
points(test$medv,pr.lm,col='blue',pch=18,cex=0.7)
abline(0,1,lwd=2)
legend('bottomright',legend=c('NN','LM'),pch=18,col=c('red','blue'))
```



Pasul 4. Validarea incrucisata

Validarea incrucisata este o alta etapa importanta in construirea unor modele predictive. Ideea de baza a acestei metode poate fi descrisa foarte simplu. Se va executa de un numar de ori urmatoarele operatii:

- Se efectueaza divizarea in seturi de antrenare si de test;

- Se estimeaza modelul retelei pe setul de date de antranament;
- Se testeaza modelul pe setul de date de test;
- Se calculeaza eraore de predictive
- Se repeta procesul de k ori

Se calculeaza apoi eroarea medie de preditie pentru toate cele k situatii.

Vom implementa o metoda de validare incrucisata folosind o bucla `for` pentru reteaua neuronala si functia `cv.gml()` din pachetul `boot` pentru modelul de regresie liniara. Pentru modelul linear avem:

```
library(boot)
set.seed(200)
lm.fit <- glm(medv~.,data=data)
cv.glm(data,lm.fit,K=10)$delta[1]
```

23.83560156

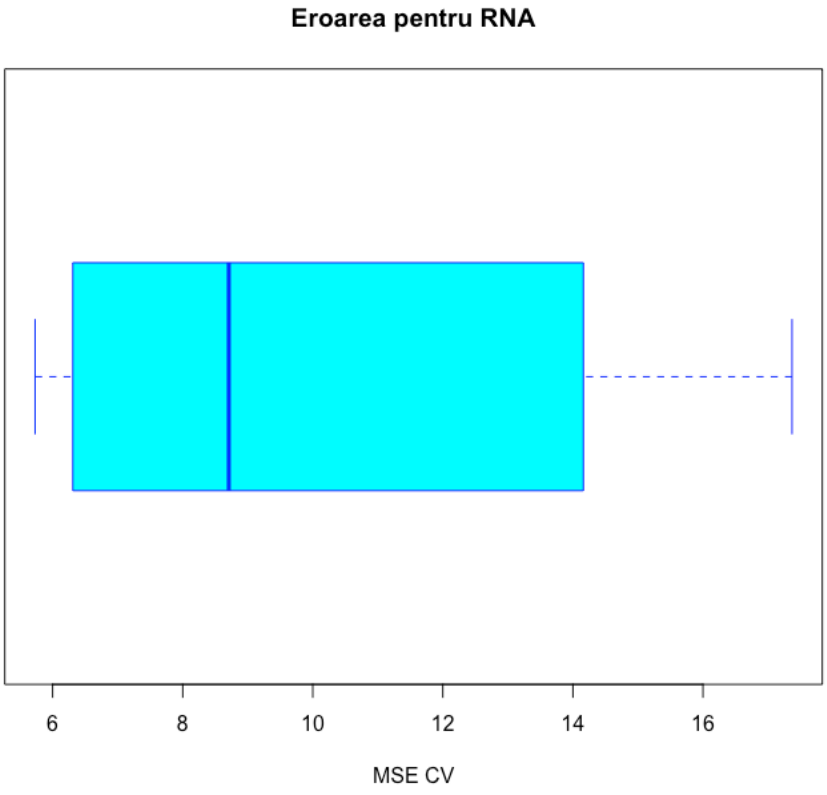
Pentru reteaua neuronala vom proceda in felul urmator. Vom diviza setul de date initial in 90% pentru antrenament si 10% pentru test de 10 ori, in mod aleator. Deoarece procesul dureaza destul de multvom utiliza un `progress-bar` utilizand pachetul plyr.

```
set.seed(450)
cv.error <- NULL
k <- 10

library(plyr)
pbar <- create_progress_bar('text')
pbar$init(k)

for(i in 1:k){
    index <- sample(1:nrow(data),round(0.9*nrow(data)))
    train.cv <- scaled[index,]
    test.cv <- scaled[-index,]

    nn <- neuralnet(f,data=train.cv,hidden=c(5,2),linear.output=T)

    pr.nn <- compute(nn,test.cv[,1:13])
    pr.nn <- pr.nn$net.result*(max(data$medv)
min(data$medv))+min(data$medv)
```

```
    test.cv.r <- (test.cv$medv)*(max(data$medv)
min(data$medv))+min(data$medv)

    cv.error[i] <- sum((test.cv.r - pr.nn)^2)/nrow(test.cv)

    pbar$step()
}
```

Calculam eroarea media si o reprezentam grafic::

```
mean(cv.error)
[1] 9.979024

cv.error
[1]  6.310575 15.769519  5.730131 10.520947  6.121161  6.389967
8.004786 17.369282  9.412778
[10] 14.161090

boxplot(cv.error,xlab='MSE CV',col='cyan', border='blue',names='CV
error (MSE)', main='Eroarea pentru RNA',horizontal=TRUE)
```



Se poate observa usor ca valoarea medie a MSE pentru reteaua neuronala este mai mica decat cea pentru modelul de regresie linear cu toate ca exista o anumite variabilitate a acesteia in functie de divizarea setului de date si de initializarea aleatoare a ponderilor retelei. Pentru o mai buna precizie se pot repeta calculele setand radacina generatorului de numere aleatoare la diferite valori.

Obervatie finala

Retele neuronale se aseamana cu o cutie neagra: explicarea output-ului este mult mai dificila decat in cazul unui model linear. In plus, trebuie luate o serie de precautii inainte de utilizarea retelei.

Tema:

Utilizand setul de date `Carseats` din pachetul `ISLR` construiti un model de regresie liniara si o retea neuronala pentru a previziona valoarea vanzarilor (variabila Sales) in functie de restul variabilelor din set. Comparati performantele celor doua modele.

Construiti diferite RNA cu unul sau doua nivele ascunse si cu un numar diferit de neuroni in nivelul (nivelele ascunse).