APA - Problemes 4 Grupal

Cesc Folch, Dean Zhu, Carles Balsells, Jordi Armengol

Novembre 2018

Problema 7 Reconeixement de lletres amb la xarxa MLP [R, G]

Prenem les lletres A, B, C, ..., Z codificades digitalment en una quadrícula de 7 x 5. Per exemple, A es codifica com:

0	1	1	1	0
1	0	0	0	1
1	0	0	0	1
1	1	1	1	1
1	0	0	0	1
1	0	0	0	1
1	0	0	0	1

L'arxiu `letters.txt` conté codificacions de 26 lletres, cadascuna representada com un vector de longitud 35. La tasca és dissenyar una xarxa neuronal que classifiqui imatges (possiblement corruptes) com a lletres de l'alfabet.

1. Dissenyeu una funció que generi versions corruptes d'una lletra, a còpia de canviar un cert número de bits de manera aleatòria. Una manera senzilla és generar primer el número de bits corruptes –p.e. amb una Poisson ($\lambda = 1.01$)– seleccionar els bits concrets (uniformement) i després invertir-los.

2. Dissenyeu una funció que, partint de les lletres netes (arxiu `letters.txt`), generi unes dades corruptes que usarem com a mostra de *training*, de mida n.

3. Entreneu una xarxa MLP per apendre la tasca. Caldrà que estimeu la millor arquitectura, cosa que podeu fer per $cross - validation$, usant regularització al vostre criteri.

4. Reporteu els resultats de predicció en una mostra de test gran –també generada per vosaltres, i de manera anàloga a la de *training*.

---

1. Dissenyeu una funció que generi versions corruptes d'una lletra, a còpia de canviar un cert número de bits de manera aleatòria. Una manera senzilla és generar primer el número de bits corruptes –p.e. amb una Poisson ($\lambda = 1.01$)– seleccionar els bits concrets (uniformement) i després invertir-los.

```
corrompreLletra <- function(lletra) {

  lletraCorrupta <- as.numeric(as.vector(lletra[1:35]))
  nBitsCorruptes <- rpois(n=1,lambda=1.01)
  while (nBitsCorruptes > 35) {
    nBitsCorruptes <- rpois(n=1,lambda=1.01)
  }
  bitsACorrompre <- sample(1:35,nBitsCorruptes)
  lletraCorrupta[bitsACorrompre] <-  as.numeric(
        as.vector(mapply((function (x) if(x==1) 0 else 1),lletraCorrupta[bitsACorrompre])))
```

```
  lletraCorrupta <- as.data.frame(c(lletraCorrupta,lletra[36]))

  return(lletraCorrupta)
}
```

En particular, la funció rep un dataframe d'una única fila amb 36 columnes, l'última de les quals és l'etiqueta que indica de quina lletra es tracta. També retorna un dataframe del mateix tipus però potencialment corrupte.

2. Dissenyeu una funció que, partint de les lletres netes (arxiu `letters.txt`), generi unes dades corruptes que usarem com a mostra de *training*, de mida n.

```
generarLletresCorruptes <- function(lletres,n) {
  if (n == 0) {
    return(data.frame())
  }
  i <- 1
  mostres <- sample(1:26,n,replace=TRUE)
  lletresCorruptes<-as.data.frame(apply(lletres[mostres[i],],1,corrompreLletra))
  lletresCorruptes <- as.data.frame(t(lletresCorruptes))

  while (i < n) {
    i <- i + 1
    aux <-as.data.frame(apply(lletres[mostres[i],],1,corrompreLletra))
    aux <- as.data.frame(t(aux))
    lletresCorruptes <- rbind(lletresCorruptes,aux)

  }
  rownames(lletresCorruptes) <- c(1:n)
  colnames(lletresCorruptes) <- c(1:35,"label")
  lletresCorruptes$label <- as.factor(lletresCorruptes$label)
  return(lletresCorruptes)
}
```

Partint del dataframe original i de la funció anterior, generem un nou dataframe amb n files potencialment corruptes.

3. Entreneu una xarxa MLP per apendre la tasca. Caldrà que estimeu la millor arquitectura, cosa que podeu fer per $cross - validation$, usant regularització al vostre criteri.

```
file <- "letters.txt"
dades <- read.table(file, header = FALSE, sep = "␣")

library(caret)
library(MASS)
library(nnet)
set.seed (4567)

learn <- generarLletresCorruptes(dades,500)

## specify 10x10 CV
trc <- trainControl (method="repeatedcv", number=10, repeats=10,verboseIter = TRUE)

(decays <- 10^seq(-3,0,by=0.05))
## WARNING: this takes some minutes
model.10x10CV <- train (label ~., data = learn, method='nnet', maxit = 300,
```

```
                trace = FALSE, metric="Accuracy",
                tuneGrid = expand.grid(.size=15,.decay=decays), trControl=trc)

save(model.10x10CV,file="problema_7_model.mod")
```

Resultats del training

```
load ("problema_7_model.mod")

## We can inspect the full results
model.10x10CV$results

## and the best model found
model.10x10CV$bestTune
```

Utilitzarem el segon mètode que vam veure al laboratori per entrenar la xarxa neuronal: un 10×10 *cross-validation* i provant amb diferents valors per *decay* havent fixat un nombre de neurones de la capa oculta relativament alt (15). Ho farem utilitzant la llibreria `caret` (`trainControl`). Noti's que *nnet* tampoc ens deixaria posar moltes neurones perquè de seguida arriba al límit de pesos (tenint en compte que hi ha 35 entrades i 26 sortides), a part que seria computacionalment molt costós. El *decay* dels pesos ens permetrà regularitzar per intentar reduir el potencial *overfitting*. Provar diversos valors també per la mida de la xarxa (el primer mètode que vam veure a laboratori) seria molt costós computacionalment i preferim fixar un valor més alt i a canvi provar *decays*. Com a màxim nombre d'iteracions per cada crida a `nnet` utilitzarem `maxit` $= 300$.

Al tenir 35 bits cada imatge, l'entrada serà de mida 35. Al ser un problema de classificació de 26 classes hi haurà 26 neurones de sortida. A classe de teoria vam veure que amb classificació amb més de 2 classes, s'usa la *softmax* com a funció d'activació de les neurones de la capa de sortida i *cross-entropy* generalitzada com a funció d'error, encara que d'aquests paràmetres ja se n'encarrega la pròpia rutina `nnet`.

Els resultats del training són que el millor *decay* és $\lambda = 0.4467$. L'*accuracy* obtinguda en les dades d'aprenentatge és de 99.944%, però per avaluar com de bo és el model resultant hauríem d'esperar als resultats del test, per veure si és capaç de generalitzar a dades potencialment noves o altrament pateix d'*overfitting*.

4. Reporteu els resultats de predicció en una mostra de test gran –també generada per vosaltres, i de manera anàloga a la de *training*.

```
test <- generarLletresCorruptes(dades,1500)
test$label <- as.factor(test$label)

p <- factor(predict (model.10x10CV, newdata=test, type="raw"),levels=dades[,36])
```

L'*accuracy* aconseguida en el test és del 99.267%. Pensem que els resultats han estat tan bons perquè si ens fixem en les dades que hem generat les lletres no són particularment corruptes. La Poisson amb $\lambda = 1.01$ en moltes ocasions genera un 0 o un 1, i en general molt pocs bits acaben corruptes. Una xarxa neuronal ben entrenada, encara que sigui d'una sola capa oculta i "només" 15 neurones, és un mètode perfectament capaç d'aprendre aquesta tasca. A més, les dades de *learning* són força grans i provenen de la mateixa distribució que les de *test*.

Podem observar que la matriu de confusió és pràcticament diagonal, cosa que indica que ha encertat en la majoria dels casos. En els casos d'error observem que hi havia una certa dificultat en el cas (lletres molt similars, sobretot en cas que hi hagi més de 2 o 3 bits deformats, com ara en el cas de les lletres O i D, O i G,). Notem també que tot i haver-se generat de forma aleatòria, tant les dades de *learning* com les de *test* estan força balancejades, és a dir, totes les classes tenen de l'ordre dels mateixos casos.

Confusion Matrix and Statistics

Prediction \ Reference	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	39	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
B	0	40	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
C	0	0	34	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
D	0	0	0	48	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
F	0	0	0	0	0	26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
G	0	0	0	0	0	0	44	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
K	0	0	0	0	0	0	0	0	0	0	45	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L	0	0	0	0	0	0	0	0	0	0	0	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0
M	0	0	0	0	0	0	0	0	0	0	0	1	33	0	0	0	0	0	0	0	0	0	0	0	0	0
N	0	0	0	0	0	0	0	0	0	0	0	0	0	47	0	0	0	0	0	0	0	0	0	0	0	0
O	0	0	0	1	0	0	1	0	0	0	0	0	0	0	32	0	0	0	0	0	0	0	0	0	0	0
P	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38	0	0	0	0	0	0	0	0	0	0
Q	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	39	0	0	0	0	0	0	0	0	0
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	0	0	0	0	0	0	0	0
S	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	45	0	0	0	0	0	0	0
T	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	32	0	0	0	0	0	0
U	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38	0	1	0	0	0
V	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	41	0	0	0	0
W	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	44	0	0	0
X	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	50	0	0
Y	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	35	0
Z	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

Overall Statistics

```
               Accuracy : 0.99
                 95% CI : (0.9817, 0.9952)
    No Information Rate : 0.05
    P-Value [Acc > NIR] : < 2.2e-16
```