APA - Entrega 5

Gerard Barrachina, Josep de Cid, Albert Ribes, Kerstin Winter

Gener 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 Possion ($\lambda = 1.01$)- seleccionar els bits concrets (uniformement) i després invertir-los.**

```
corrupt <- function(bits) {
  rand.corrupt <- min(rpois(1, lambda = 1.01), 35)
  change.pos <- sample(35, size = rand.corrupt)
  mask <- rep(0, times = 35)
  mask[change.pos] <- 1
  changed.bits <- xor(as.numeric(bits[1:35]), mask)
  corrupted <- c(as.numeric(changed.bits), bits[36])
  return(corrupted)
}
```

Aquesta funció rebrà com a paràmetre el vector de bits que conformen una observació. Comencem generant el nombre de bits aleatoris a canviar `n`, amb una distribució Poisson amb $\lambda = 1.01$, i creant una mostra aleatòria corresponent a les `n` posicions a invertir.

Creem una màscara de bits a 0 excepte les posicions marcades anteriorment, que les posem a 1. Apliquem una *xor* entre els bits d'entrada i la màscara que em creat, de forma que s'intercanviaran 0 i 1 a les posicions on la màscara val 1.

Finalment retornem l'aplicació de la *xor* en forma numèrica, ja que la porta retorna un resultat booleà unit amb la representació de la lletra.

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

```
generateCorrupt <- function(df, n) {
  idx <- sample(1:nrow(df), n, replace = TRUE)
  new.data <- t(apply(df[idx,], MARGIN = 1, FUN = corrupt))
  new.data <- data.frame(new.data)
  rownames(new.data) <- seq(1:n)
  colnames(new.data) <- c(1:35, "letter")
  new.data$letter <- as.factor(new.data$letter)
  return(new.data)
}
```

Aquesta funció rebrà com a paràmetres el *dataframe* que conté les dades del fitxer `letters.txt`, i el nombre de mostres desitjades, parametritzat per a ser usat genèricament tant pel *training set* com pel *test set*.

En primer lloc generem unes mostres aleatòries amb la funció *sample*, corresponents a un índex del vector de bits de cada lletra (amb el paràmetre `replace = TRUE` ja que molt probablement, el paràmetre N serà superior al nombre de lletres).

Fem us de la funció *apply* per aplicar la funció *corrupt* a cada observació, accedint-hi pels índex generats al principi de la funció.

Finalment reanomenem files i columnes per simplificar el tractament i tornem els nous conjunts en forma de *dataframe*.

3. **Entreneu una xarxa MLP per aprendre la tasca. Caldrà que estimeu la millor arquitectura, cosa que podeu fer per *cross-validation*, usant regularització.**

```
training.set <- generateCorrupt(letters.df, 1500)
test.set <- generateCorrupt(letters.df, 500)

sizes <- seq(1, 15, by = 2)
decays <- 10^seq(-3, 0, by = 0.2)
trc <- trainControl(method = 'repeatedcv',
                    number = 10,
                    repeats = 5)
```

```
nnet.model <- train(letter ~ .,
                    data = training.set, trace = FALSE,
                    method = 'nnet', metric = 'Accuracy',
                    trControl = trc, maxit = 500,
                    tuneGrid = expand.grid(.size = sizes,
                                           .decay = decays))
(best <- nnet.model$bestTune)
```

En primer lloc generem els conjunts de *training* i *test*, que ens servir.

Creem un objecte de tipus *trainControl* per indicar com realitzar l'entrenament de la xarxa neuronal, amb el mètode *repeatedcv*, 10 divisions i 5 repeticions. Això farà 5 *10-fold-crossvalidations*.

Només ens cal entrenar el model, amb la funció train, definint la fórmula on *letter* és modelat respecte a la resta de variables, fent ús del mètode *nnet*, mètrica *Accuracy* i l'objecte de *trainControl* definit anteriorment. Definim també l'argument `maxit = 500` augmentant el nombre d'iteracions de l'algorisme, que per defecte és 100.

Fem servir també una grid d'hiperparàmetres, per a que el model train trobi el millor tamany i decay per a la xarxa, amb mida senar entre 1 i 15 i decays definits amb 10^x on x va de -3 a 0 avançant en passos de 0.2.

```
Resampling: Cross-Validated (10 fold, repeated 5 times)
Summary of sample sizes: 1347, 1350, 1351, 1349, 1351, 1353, ...
Resampling results across tuning parameters:

size  decay        Accuracy   Kappa
   1    0.001000000  0.3898181  0.36313461
   1    0.001584893  0.3870445  0.36004305
   1    0.002511886  0.3928940  0.36621482
 ...
   1    1.000000000  0.1196165  0.07708688
   3    0.001000000  0.9260876  0.92306257
 ...
  13    1.000000000  0.9933221  0.99304850
  15    0.001000000  0.9906623  0.99028007
 ...
  15    0.003981072  0.9930746  0.99279091
  15    0.006309573  0.9930822  0.99279914
  15    0.010000000  0.9926609  0.99236063
  15    0.015848932  0.9930723  0.99278872
  15    0.025118864  0.9931886  0.99290950
  15    0.039810717  0.9942576  0.99402268
  15    0.063095734  0.9946591  0.99444047
  15    0.100000000  0.9949360  0.99472856
  15    0.158489319  0.9947919  0.99457857
  15    0.251188643  0.9947937  0.99458074
  15    0.398107171  0.9937220  0.99346472
  15    0.630957344  0.9933383  0.99306574
  15    1.000000000  0.9941295  0.99388915

Accuracy was used to select the optimal model using the largest value.
The final values used for the model were size = 15 and decay = 0.1.
```

Observant l'objecte resultant *nnet.model*, veiem que l'arquitectura del millor model té 15 neurones en la capa oculta, aplicant una regularització amb `decay = 0.1`. Amb aquesta arquitectura, aconseguim una precisió de **0.9949360** amb les dades d'entrenament.

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

```
pred <- predict(nnet.model, newdata = test.set)
(conf <- confusionMatrix(test.set$letter, pred))
```

Creant una mostra de test amb 500 observacions, predient els resultats amb l'arquitectura trobada en l'apartat anterior, veiem que la precisió és de **0.994**, gairebé perfecta. Com a curiositat, observant la matriu de confusió, veiem que l'error obtingut és entre lletres visualment semblants, que al canviar bits, es poden confondre, com D-O o F-P (en altres execucions que s'han realitzat).

	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	26	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	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
C	0	0	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
D	0	0	1	19	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
E	0	0	0	0	17	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	15	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	13	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	18	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	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
J	0	0	0	0	0	0	0	0	0	25	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	20	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	19	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	0	15	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	17	0	0	0	0	0	0	0	0	0	0	0	0
O	0	0	0	0	0	0	0	0	0	0	0	0	0	0	21	0	0	0	0	0	0	0	0	0	0	0
P	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	18	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	25	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	23	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	18	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	27	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	16	0	0	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	15	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	15	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	27	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	21	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	23