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

Gerard Barrachina Josep de Cid Albert Ribes
Kerstin Winter

23 de diciembre de 2017

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

$$
\begin{array}{c}
0\ 1\ 1\ 1\ 0 \\
1\ 0\ 0\ 0\ 1 \\
1\ 0\ 0\ 0\ 1 \\
1\ 1\ 1\ 1\ 1 \\
1\ 0\ 0\ 0\ 1 \\
1\ 0\ 0\ 0\ 1 \\
1\ 0\ 0\ 0\ 1
\end{array}
$$

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.

```
corrupt = function(bit_vector, nchanges) {
    changes = sample(length(bit_vector), nchanges)
    g = rep(0, length(bit_vector))
    for (n in changes) {
        g[n] = 1
    }
    new = as.numeric(xor(bit_vector, g))
    return(new)
}
```

`bit_vector` es un vector que contiene 1 y 0, y `nchanges` es un entero que contiene la cantidad de bits aleatorios que deben cambiar.

Con la función `sample` decidimos qué subconjunto de bits son los que tienen que cambiar, y entonces generamos un vector binario nuevo en el que todos los elementos son 0 excepto los de las posiciones que tienen que cambiar, que serán 1. Esto lo hacemos con el bucle.

Después solo hay que hacer una `xor` entre el vector de entrada y el que hemos generado para conseguir el vector con los cambios realizados.

Finalmente retornamos el vector que hemos generado.

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.

```
generate_corrupt = function(df, n) {
    nm = c(1:35, "letter")
    g = sample(1:nrow(df), n, replace = TRUE)
    ch = rpois(n, 1.01)
    new_data = data.frame()
    for (i in 1:length(g)) {
        elem = g[i]
        new_vector = corrupt(df[elem, -36], ch[i])
        newrow = c(new_vector, df[elem, 36])
        new_data = rbind(new_data, newrow)
        new_data[i,36] = as.character(df[elem, 36])
    }
    colnames(new_data) = nm
    new_data$letter = as.factor(new_data$letter)
    return(new_data)
}
```

`df` es el dataframe que resulta de leer `letters.txt`, y `n` es un entero que indica el tamaño que debe tener la muestra de *training*.

Usamos la función `sample` para decidir de qué letras estará formado nuestro dataset, pero con `replace = TRUE`, puesto que `n` puede ser más grande que el dataframe de entrada y queremos que se puedan repetir.

Con `rpois(n, 1.01)` decidimos cuántos cambios tendrá cada letra de nuestra muestra.

Luego hacemos un bucle generando cada una de las letras corruptas usando la función del apartado anterior, y las guardamos todas en un nuevo dataframe, que es el que retornamos.

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

```
train_data = generate_corrupt(mydata, 1000)
test_data = generate_corrupt(mydata, 300)

trc <- trainControl (method="repeatedcv", number=10, repeats=5)
## WARNING: this takes maaaaaany minutes
caret.nnet.model <- train (
    letter ~.,
    data = train_data,
    method='nnet',
    metric = "Accuracy",
```

```
    trControl=trc)
test_results = predict(caret.nnet.model, test_data)
confusionMatrix(test_data$letter, test_results)
```

Usamos la biblioteca `caret` para estimar la mejor arquitectura de la red. Hacemos 5 veces $10-fold-crossvalidation$. Lo hacemos solo 5 veces porque con esta cantidad de datos el proceso tarda un rato. La métrica usada para decidir la mejor arquitectura es `Accuracy`, puesto que parece adecuada para el problema.

Después de un rato, se llega a la conclusión de que el mejor modelo tiene 5 neuronas en la capa oculta y se ha aplicado regulación con un `decay = 0.1`. Con esta arquitectura se consigue una `Accuracy` de 0,918 en los datos de *training*

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

   Hemos testeado con una muestra de 300 datos usando la arquitectura encontrada con `caret`. El resultado ha sido una `Accuracy` de 0,886, que parece bastante bien.

   Viendo la `confusionMatrix` se puede ver que muchos de los errores cometidos son con letras que visualmente son parecidas. Por ejemplo, la `D` con la `O` o la `P` con la `R`