L’algoritmo di back propagation

Vi sono vari algoritmi adottabili nell’approccio supervisionato per la modifica dei pesi.

Tra tutti, il back propagation è l’algoritmo più usato per modificare i valori sinaptici e si può utilizzare indipendentemente dall’architettura adottata e dal numero di strati intermedi.

Il principio dell’algoritmo è la modifica dei pesi delle connessioni condotta in modo da minimizzare una certa funzione errore **E**, che dipende dal vettore $\mathbf{o_i}$ restituito dalla rete per l’ingresso $\mathbf{x_i}$, e dal vettore di output $\mathbf{y_i}$ che desideriamo ottenere (e che fa parte del training set).
Il training set è l’insieme di n coppie di vettori (x_i,y_i), con i=1, …, n.

Considerato k l'indice che rappresenta il valore corrispondente al k-simo neurone di output, la funzione errore da minimizzare sarà:

$$E(w) = \frac{1}{2} \sum_i \sum_k (o_k^i - y_k^i)^2$$

E(w) dipende dai pesi, che variano nel tempo, e per minimizzarla si può utilizzare l'algoritmo della discesa del gradiente (**gradient-descent**).

Il gradient-descent parte da un punto generico x_0 e calcola il gradiente $\nabla f(x_0)$, che fornisce la direzione verso la quale muoversi e lungo la quale si ha il massimo incremento (decremento se si considera $-\nabla$).
Una volta definita la direzione, ci si muove di una distanza η predefinita, trovando un nuovo punto x_1 su cui viene calcolato di nuovo il gradiente.

Si continua così, finché il gradiente risulta nullo.

Di fatto, l'algoritmo di back propagation è diviso in due passi:

- **forward pass**: l'input fornito alla rete viene propagato ai livelli successivi (il flusso va in avanti – forward). Si calcola l'errore commesso E(w)
- **backward pass**: l'errore E(w) è propagato all'indietro (backward) ed i pesi vengono aggiornati opportunamente.

Vediamo quali sono i passi da compiere per addestrare la rete utilizzando l'approccio supervisionato:

- creare l'insieme di input pattern ed il corrispondente output pattern desiderato
- inizializzare i pesi della rete a valori casuali piccoli e a norma nulla (valore assoluto della radice quadrata della somma dei quadrati dei componenti dei vettori)

- iniziare il ciclo di apprendimento (che terminerà solo quando l'errore risulta minore della soglia che si è prefissata o dopo un numero di iterazioni deciso a priori)
  - feedforward (da input a output)
  - si estrae (ad esempio a caso) un input pattern tra quelli selezionati. Si ricordi che per ottenere un buon addestramento, l'insieme dei pattern di input deve essere completo. *(continua)*

    Dunque, scelto un pattern di input, nell'estrazione di quello nuovo, il precedente non deve partecipare alla scelta. In questo modo si considereranno solo gli input non ancora estratti, per poi ripescare dall'insieme completo quando tutti sono stati estratti almeno una volta. Si procede in modo simile anche se si sceglie di adottare un meccanismo k-fold.
    - si calcola il valore di tutti i neuroni successivi (con la sommatoria di produttorie che conosciamo, ormai, bene)

    - si detrae dal risultato il valore di soglia di attivazione del neurone (in alternativa, come abbiamo visto, è possibile simulare la soglia con l'aggiunta di un neurone ad ingresso fisso di valore pari a 1)
    - si filtra l'uscita del neurone e si riadatta il ciclo facendo diventare tale valore l'input del neurone successivo
- si confronta il risultato ottenuto dalla rete con l'output pattern corrispondente all'input appena utilizzato e si ricava l'errore attuale della rete
- back propagation (da output a input)
    - si calcola la correzione da apportare ai pesi secondo la regola scelta
    - si applica la correzione ai pesi dello strato

Link utile per approfondire ulteriormente l'argomento reti neurali:

http://stor.altervista.org/informatica/nn/nn.htm (ANN affrontate in modo pratico)

Come creare una rete neurale

Per creare una rete neurale è possibile utilizzare

- **sw di simulazione di ANN già pronti** (vedremo nel seminario sulle ANN). Sconsigliato, perché anche per i neofiti è difficile capire bene come utilizzare questi programmi poiché sono "a scatola chiusa". Vi sono sia realizzazioni a pagamento, non economiche ma almeno con buoni manuali, sia sw più economici o a volte gratuiti ma con il difetto di non essere corredati da help o da un buon manuale. Alcuni esempi (e link) potrete trovarli qui:
  http://www.msandri.it/complex/neural/softwa~1.htm
  Un ambiente free piuttosto semplice è JOONE Editor, cui daremo uno sguardo veloce. *(continua)*
- **linguaggi di programmazione**. Per i più esperti (in linguaggi di programmazione, intendo) è possibile programmare ex novo una rete neurale utilizzando il linguaggio preferito (C++, Java, ecc.). La rete neurale in sé non comporta molte righe di codice ma, ovviamente, occorre conoscere il linguaggio di programmazione.

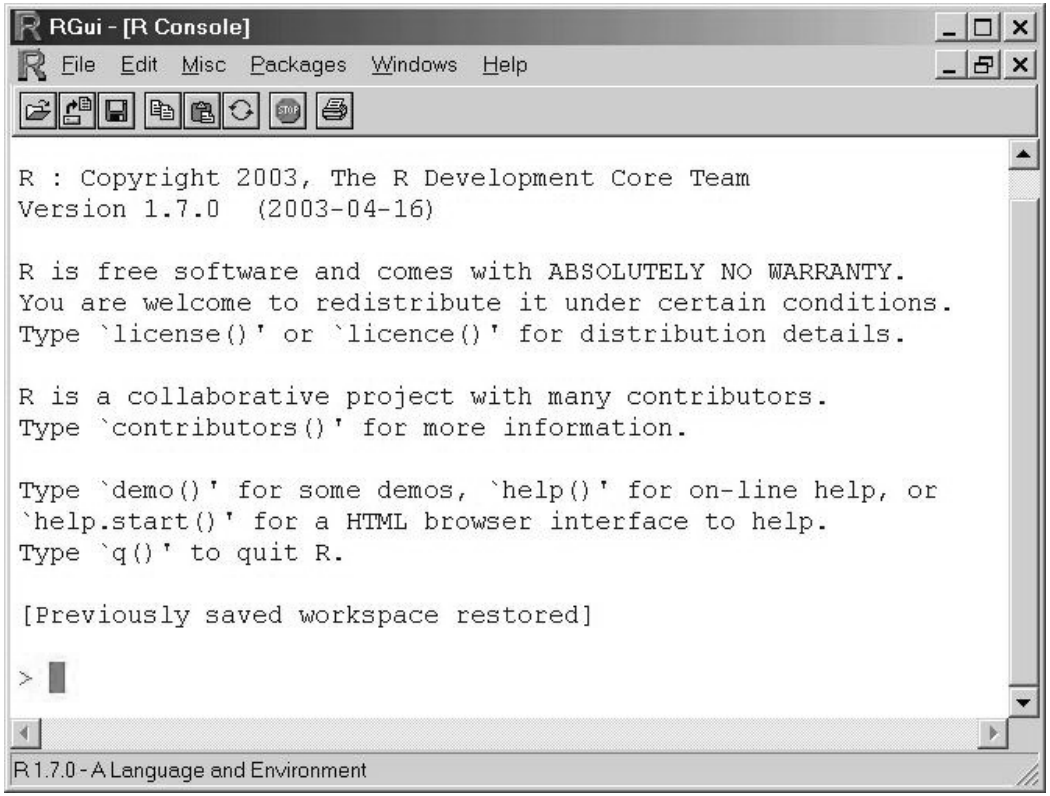
- **Nuove implementazioni sviluppate in ambienti statistici**. Tra i più conosciuti Matlab (che però non è free) e R (opensource e free). Contengono package (librerie) che permettono di sviluppare facilmente una rete neurale. Nell'intento di farvi cosa gradita (visto che è utile per molte applicazioni) vedremo R e come sviluppare una rete semplice in questo ambiente. Anche Mathematica è un ambiente semplice da utilizzare.

Creare una rete neurale con R

R è un software opensource, free, molto potente nato per implementazioni statistiche ma contenente molti package (librerie) per applicazioni diverse. Le librerie sono insiemi di procedure e funzioni accessibili da un comando (sintassi) diretto per realizzare un'applicazione. Qui: http://www.cran.r-project.org/ trovate un elenco completo di tutte le librerie presenti in R.
È possibile scaricare R da questo sito http://www.r-project.org/ in cui sono presenti anche manuali, documentazione e link al wiki per chi intende ampliare le proprie conoscenze. I manuali di R presenti sul sito sono fatti molto bene, ma sono in Inglese. Per chi preferisse l'Italiano, ottimi manuali di introduzione ad R in Rete possono essere reperiti qui:
http://torinor.net/download/EnnioDispense.pdf
http://cran.r-project.org/doc/contrib/Mineo-dispensaR.pdf
Noi vedremo le basi di R, giusto per poter capire come si utilizza l'ambiente e come sviluppare una rete neurale semplice di cui faremo l'esempio nel seguito. Per gli approfondimenti, si rimanda ai link sopra riportati.

Primi passi in R

(tratto da http://math.illinoisstate.edu/dhkim/rstuff/rtutor.html)



Una volta scaricata e installata la versione di R adatta al proprio sistema operativo, basta avviare R dall'icona sul desktop oppure dal menu di avvio per visualizzare la schermata della console:
Come uscire da R? Basta scrivere `q( )` al prompt (>) e premere invio. Apparirà una dialog box che richiede se si desidera salvare ciò che si è creato in R per vederlo apparire nuovamente alla prossima sessione di R.

I comandi scritti sulla console possono essere sempre facilmente richiamati e modificati agendo sulle frecce posizionate sulla tastiera.

Vediamo subito due comandi utili:

```
> objects() # list the names of all objects
> rm(data1) #remove the object named data1 from the current environment
```

Operazioni di base

Ovviamente in R è possibile innanzitutto compiere tutte le operazioni di base, quali, ad esempio:

```
> 2 + 3 * 5 # Note the order of operations.
> log (10) # Natural logarithm with base e=2.718282
> 4^2 # 4 raised to the second power
> 3/2 # Division
> sqrt (16) # Square root
> abs (3-7) # Absolute value of 3-7
> pi # The mysterious number
> exp(2) # exponential function
> 15 %/% 4 # This is the integer divide operation
> # This is a comment line
```

Vi sono poi gli operatori di assegnazione (<-) che immagazzinano il valore (object), o il risultato dell'espressione, posto sulla parte destra di <- alla variabile posta a sinistra. Una volta effettuata l'assegnazione, la variabile può essere utilizzata come il valore stesso. Attenzione: R è case sensitive. Prima di assegnare un nome, assicuratevi che non sia già utilizzato da R nei molti pacchetti che mette a disposizione. E' sufficiente scrivere il nome scelto e dare invio: se R ritorna un errore, vuol dire che il nome che avete scelto non è in uso. Vediamo alcuni esempi di assegnazione:

```
> x<- log(2.843432) *pi
> x
[1] 3.283001
> sqrt(x)
[1] 1.811905
> floor(x) # largest integer less than or equal
to x (Gauss number)
[1] 3
```

```
> ceiling(x) # smallest integer greater than or
equal to x
[1] 4
```
R può anche gestire numeri complessi:
```
> x<-3+2i
> Re(x) # Real part of the complex number x
[1] 3
> Im(x) # Imaginary part of x
[1] 2
> y<- -1+1i
> x+y
[1] 2+3i
> x*y
[1] -5+1i
```

Vettori

R gestisce gli oggetti vettore in modo semplice ed intuitivo:

```
> x<-c(1,3,2,10,5) #create a vector x with 5 components
> x
[1] 1 3 2 10 5
> y<-1:5 #create a vector of consecutive integers
> y
[1] 1 2 3 4 5
> y+2 #scalar addition
[1] 3 4 5 6 7
> 2*y #scalar multiplication
[1] 2 4 6 8 10
> y^2 #raise each component to the second power
```

```
[1] 1 4 9 16 25
> 2^y #raise 2 to the first through fifth
power
[1] 2 4 8 16 32
> y #y itself has not been unchanged
[1] 1 2 3 4 5
> y<-y*2
> y #it is now changed
[1] 2 4 6 8 10
```

Alcuni esempi di operazioni sui vettori:

```
> x<-c(1,3,2,10,5); y<-1:5 #two or more
statements are separated by semicolons
> x+y
[1] 2 5 5 14 10
> x*y
[1] 1 6 6 40 25
> x/y
[1] 1.0000000 1.5000000 0.6666667 2.5000000
1.0000000
> x^y
[1] 1 9 8 10000 3125
> sum(x) #sum of elements in x
[1] 21
> cumsum(x) #cumulative sum vector
[1] 1 4 6 16 21
```

```
> diff(x) # first difference
[1] 2 -1 8 -5
> diff(x,2) #second difference
[1] 1 7 3
> max(x) #maximum
[1] 10
> min(x) #minimum
[1] 1
```

L'ordinamento dei dati può essere effettuato utilizzando il commando `sort()`:

```
> x
[1] 1 3 2 10 5
> sort(x) # increasing order
[1] 1 2 3 5 10
> sort(x, decreasing=T) # decreasing order
[1] 10 5 3 2 1
```

Inoltre estrarre i componenti di un vettore per effettuare operazioni è estremamente semplice:

```
> x
[1] 1 3 2 10 5
> length(x) # number of elements in x
[1] 5
> x[3] # the third element of x
[1] 2
> x[3:5] # the third to fifth element of x,
inclusive
[1] 2 10 5
> x[-2] # all except the second element
[1] 1 2 10 5
> x[x>3] # list of elements in x greater than
3
[1] 10 5
```

Si possono anche avere vettori logici:

```
> x>3
[1] FALSE FALSE FALSE TRUE TRUE
> as.numeric(x>3) # as.numeric() function
coerces logical components to numeric
[1] 0 0 0 1 1
> sum(x>3) # number of elements in x greater
than 3
[1] 2
> (1:length(x))[x<=2] # indices of x whose
components are less than or equal to 2
[1] 1 3
> z<-as.logical(c(1,0,0,1)) # numeric to
logical vector conversion
> z
[1] TRUE FALSE FALSE TRUE
```

Vettori di caratteri:

```
> colors<-c("green", "blue", "orange", "yellow",
"red")
> colors
[1] "green" "blue" "orange" "yellow" "red"
```

Inoltre è possibile assegnare nomi a component individuali e poi riferirsi ad esse tramite il nome, come negli esempi che seguono.

```
> names(x) # check if any names are attached to x
NULL
> names(x)<-colors # assign the names using the
character vector colors
> names(x)
[1] "green" "blue" "orange" "yellow" "red"
> x
green blue orange yellow red
1 3 2 10 5
```

```
> x["green"] # component reference by its
name
green
1
> names(x)<-NULL # names can be removed by
assigning NULL
> x
[1] 1 3 2 10 5
```

`seq()` and `rep()` provide convenient ways to a construct vectors with a certain pattern.

```
> seq(10)
[1] 1 2 3 4 5 6 7 8 9 10
> seq(0,1,length=10)
[1] 0.0000000 0.1111111 0.2222222 0.3333333
0.4444444 0.5555556 0.6666667
[8] 0.7777778 0.8888889 1.0000000
```

```
> seq(0,1,by=0.1)
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9
1.0
> rep(1,3)
[1] 1 1 1
> c(rep(1,3),rep(2,2),rep(-1,4))
[1] 1 1 1 2 2 -1 -1 -1 -1
> rep("Small",3)
[1] "Small" "Small" "Small"
> c(rep("Small",3),rep("Medium",4))
[1] "Small" "Small" "Small" "Medium" "Medium"
"Medium" "Medium"
> rep(c("Low","High"),3)
[1] "Low" "High" "Low" "High" "Low" "High"
```

Matrici

Come sapete, una matrice è un array di righe e colonne. Uno dei modi più facili di creare una matrice consiste nel combinare vettori di uguale lunghezza utilizzando il comando cbind(), che sta per "column bind":

```
> x
[1] 1 3 2 10 5
> y
[1] 1 2 3 4 5
> m1<-cbind(x,y);m1
x y
[1,] 1 1
[2,] 3 2
[3,] 2 3
[4,] 10 4
[5,] 5 5
```

```
> t(m1) # transpose of m1
[,1] [,2] [,3] [,4] [,5]
x 1 3 2 10 5
y 1 2 3 4 5
> m1<-t(cbind(x,y)) # Or you can combine them
and assign in one step
> dim(m1) # 2 by 5 matrix
[1] 2 5
> m1<-rbind(x,y) # rbind() is for row bind
and equivalent to t(cbind()).
```

Naturalmente gli elementi della matrice si possono elencare in modo diretto:

```
> m2<-matrix(c(1,3,2,5,-1,2,2,3,9),nrow=3);m2
[,1] [,2] [,3]
[1,] 1 5 2
[2,] 3 -1 3
[3,] 2 2 9
```

Si noti che gli elementi sono utilizzati per “riempire” la prima Colonna, quindi la sseconda colonna e così via. Per riempire la matrice per righe, è sufficiente utilizzare l’opzione `byrow=T`:

```
> m2<-matrix(c(1,3,2,5,-
1,2,2,3,9),ncol=3,byrow=T);m2
[,1] [,2] [,3]
[1,] 1 3 2
[2,] 5 -1 2
[3,] 2 3 9
```

Per estrarre le componenti da una matrice è necessario specificare uno o più indici:

```
> m2
[,1] [,2] [,3]
[1,] 1 3 2
[2,] 5 -1 2
[3,] 2 3 9
> m2[2,3] #element of m2 at the second row,
third column
[1] 2
> m2[2,] #second row
[1] 5 -1 2
> m2[,3] #third column
[1] 2 2 9
```

```
> m2[-1,] #submatrix of m2 without the first
row
[,1] [,2] [,3]
[1,] 5 -1 2
[2,] 2 3 9
> m2[,-1] #ditto, sans the first column
[,1] [,2]
[1,] 3 2
[2,] -1 2
[3,] 3 9
> m2[-1,-1] #submatrix of m2 with the first
row and column removed
[,1] [,2]
[1,] -1 2
[2,] 3 9
```

Inoltre è possibile effettuare operazioni componente per componente:

```
> m1<-matrix(1:4, ncol=2); m2<-
matrix(c(10,20,30,40),ncol=2)
> 2*m1 # scalar multiplication
[,1] [,2]
[1,] 2 6
[2,] 4 8
> m1+m2 # matrix addition
[,1] [,2]
[1,] 11 33
[2,] 22 44
> m1*m2 # component-wise multiplication
[,1] [,2]
[1,] 10 90
[2,] 40 160
```

Si noti che m1*m2 non è la moltiplicazione usuale, per la quale, invece, si utilizza l'operatore %*%

```
> m1 %*% m2
[,1] [,2]
[1,] 70 150
[2,] 100 220
> solve(m1) #inverse matrix of m1
[,1] [,2]
[1,] -2 1.5
[2,] 1 -0.5
> solve(m1)%*%m1 #check if it is so
[,1] [,2]
[1,] 1 0
[2,] 0 1
> diag(3) #diag() is used to construct a k by
```

```
k identity matrix
[,1] [,2] [,3]
[1,] 1 0 0
[2,] 0 1 0
[3,] 0 0 1
> diag(c(2,3,3)) #as well as other diagonal
matrices
[,1] [,2] [,3]
[1,] 2 0 0
[2,] 0 3 0
[3,] 0 0 3
```

Per altre operazioni più complesse, si rimanda alla documentazione presente sul sito di R.

I Data frame

I Data frame sono array di dati le cui colonne possono contenere valori di tipo diverso (numeri, caratteri, ecc.) Data frame di piccole dimensioni possono essere creati con la funzione `data.frame()`. Ecco un data frame di dati di automobili:

Make	**Model**	**Cylinder**	**Weight**	**Mileage**	**Type**
Honda	Civic	V4	2170	33	Sporty
Chevrolet	Beretta	V4	2655	26	Compact
Ford	Escort	V4	2345	33	Small
Eagle	Summit	V4	2560	33	Small
Volkswagen	Jetta	V4	2330	26	Small
Buick	Le Sabre	V6	3325	23	Large
Mitsubishi	Galant	V4	2745	25	Compact
Dodge	Grand Caravan	V6	3735	18	Van
Chrysler	New Yorker	V6	3450	22	Medium
Acura	Legend	V6	3265	20	Medium

Vediamo come creare il dataframe:

```
> Make<-
c("Honda","Chevrolet","Ford","Eagle","Volkswa
gen","Buick","Mitsbusihi",
+ "Dodge","Chrysler","Acura")
> Model<-
c("Civic","Beretta","Escort","Summit","Jetta"
,"Le Sabre","Galant",
+ "Grand Caravan","New Yorker","Legend")
```

Il segno + presente nelle righe sopra, viene inserito automaticamente da R quando viene dto l'invio senza aver completato la lista. Inoltre il commando `rep()` serve per far ripetere più volte il comando a R. Ad esempio, `rep("V4",5)` dice ad R di ripetere V4 per cinque volte.

```
> Cylinder<-
c(rep("V4",5),"V6","V4",rep("V6",3))
> Cylinder
[1] "V4" "V4" "V4" "V4" "V4" "V6" "V4" "V6"
"V6" "V6"
> Weight<-
c(2170,2655,2345,2560,2330,3325,2745,3735,345
0,3265)
> Mileage<-c(33,26,33,33,26,23,25,18,22,20)
> Type<-
c("Sporty","Compact",rep("Small",3),"Large","
Compact","Van",rep("Medium",2))
```

La funzione `data.frame()` combina i sei vettori in un singolo data frame

```
> Car<-
data.frame(Make,Model,Cylinder,Weight,Mileage
,Type)
> Car
Make Model Cylinder Weight Mileage Type
1 Honda Civic V4 2170 33 Sporty
2 Chevrolet Beretta V4 2655 26 Compact
3 Ford Escort V4 2345 33 Small
4 Eagle Summit V4 2560 33 Small
5 Volkswagen Jetta V4 2330 26 Small
6 Buick Le Sabre V6 3325 23 Large
7 Mitsbusihi Galant V4 2745 25 Compact
8 Dodge Grand Caravan V6 3735 18 Van
```

```
9 Chrysler New Yorker V6 3450 22 Medium
10 Acura Legend V6 3265 20 Medium
> names(Car)
[1] "Make" "Model" "Cylinder" "Weight"
"Mileage" "Type"
```

Esattamente come negli oggetti matrice, le informazioni parziali possono essere facilmente estratte dai data frame:

```
> Car[1,]
Make Model Cylinder Weight Mileage Type
1 Honda Civic V4 2170 33 Sporty
```

Inoltre, single colonne possono essere richiamate attraverso le loro etichette:

```
> Car$Mileage
[1] 33 26 33 33 26 23 25 18 22 20
> Car[,5] #equivalent expression, less
informative
> mean(Car$Mileage) #average mileage of the
10 vehicles
[1] 25.9
> min(Car$Weight)
[1] 2170
```

Il comando table() fornisce una tabella di frequenza:

```
> table(Car$Type)
Compact Large Medium Small Sporty Van
2 1 2 3 1 1
```

Se si desidera ordinare I dati secondo il peso del veicolo, occorre usare il commando `order(`

```
> i<-order(Car$Weight);i
[1] 1 5 3 4 2 7 10 6 9 8
> Car[i,]
Make Model Cylinder Weight Mileage Type
1 Honda Civic V4 2170 33 Sporty
5 Volkswagen Jetta V4 2330 26 Small
3 Ford Escort V4 2345 33 Small
4 Eagle Summit V4 2560 33 Small
2 Chevrolet Beretta V4 2655 26 Compact
7 Mitsbusihi Galant V4 2745 25 Compact
10 Acura Legend V6 3265 20 Medium
6 Buick Le Sabre V6 3325 23 Large
9 Chrysler New Yorker V6 3450 22 Medium
8 Dodge Grand Caravan V6 3735 18 Van
```

Come modificare i data objects

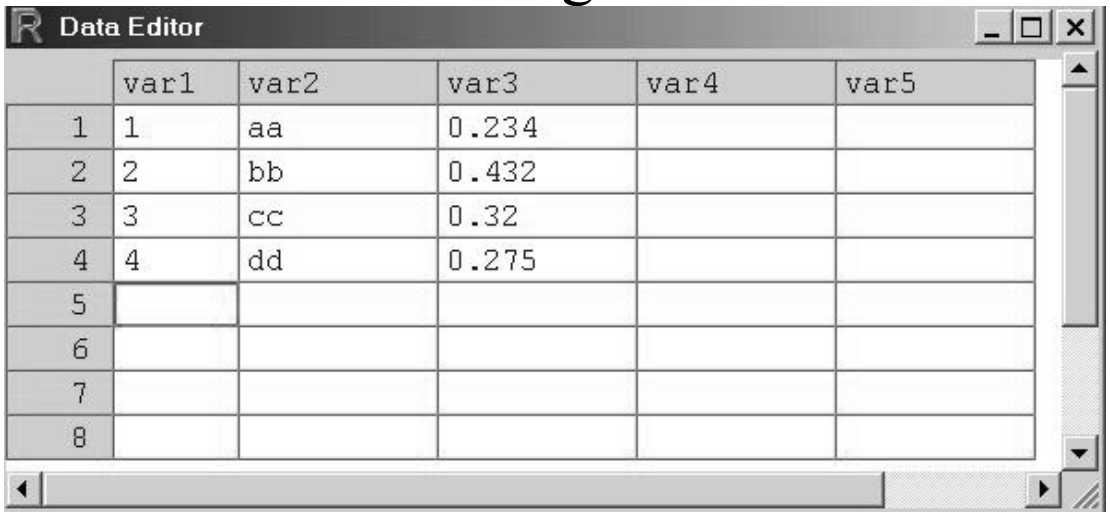
```
> y
[1] 1 2 3 4 5
```

Se si desidera modificare I data object, è necessario utilizzare la funzione edit() e assengarla ad un oggetto. Ad esempio il commando che segue apre notepad per modificare l'oggetto-

```
> y<-edit(y)
```

Se invece si preferisce inserire i dati di un data frame un stile spreadsheet, occorre utilizzare la sintassi che segue.

`> data1<-edit(data.frame())` In questo caso apparirà una schermata come in figura:



	var1	var2	var3	var4	var5
1	1	aa	0.234		
2	2	bb	0.432		
3	3	cc	0.32		
4	4	dd	0.275		
5					
6					
7					
8					

E’ possibile anche cambiare il nome della variabile facendo click una volta sulla cella che la contiene, aprendo così una dialog box come la seguente:

Una volta finito, basta un click su  per tornare alla finestra Data Editor. Chiudendo il Data Editor si torna alla finestra di comando R (R Console). Per verificare che i cambiamenti siano avvenuti, basta scrivere:

```
> data1
```

R mette a disposizione molti tipi di grafici (nelle dispense trovate tutte le indicazioni)

Anche per l'analisi statistica R mette a disposizione molte funzioni e troverete degli esempi sempre nelle dispense presenti sul sito.

Nelle dispense sono presenti, inoltre vari altri comandi, compresi quelli che consentono l'import e l'esport di dati.

Consultare l'help

Per default, R mette a disposizione, al suo interno, alcuni manuali in pdf fatti molto bene.
Per accedere al manual, si faccia click su Help | Manuals.
Per usare invece l'help per avere informazioni su una specifica funzione o comando, si usi il comando help()

```
> help(read.table)
```

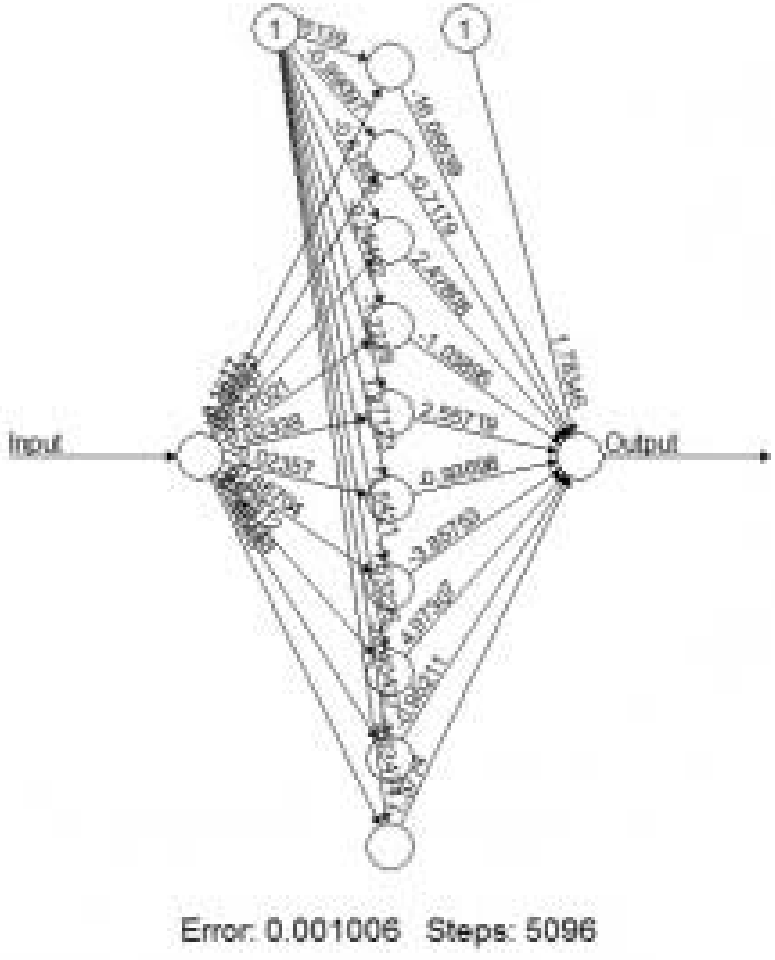
Esempio di rete neurale in R

In questo esempio viene sviluppata una rete neurale (Multilayer perceptron) in grado prendere un numero e calcolare la radice quadrata con la maggior precisione possibile.
Per addestrare e costruire la rete neurale sarà utilizzata la libreria R "neuralnet".

La rete prodotta con questo esempio è quella mostrata nell'immagine.

Prende un singolo input (il numero di cui si desidera la radice quadrata) e produce un singolo output (la radice quadrata dell'input).

Nel mezzo dell'immagine vedete 10 hidden neuron che saranno addestrati.



L’output risulterà così:

Input	Output Atteso	Neural Net Output
1	1	0.9623402772
4	2	2.0083461217
9	3	2.9958221776
16	4	4.0009548085
25	5	5.0028838579
36	6	5.9975810435
49	7	6.9968278722
64	8	8.0070028670
81	9	9.0019220736
100	10	9.9222007864

Come si può vedere, la rete neurale svolge abbastanza bene il compito perché l’errore più grande è nel calcolo della radice quadrata di 1 e si aggira intorno al 4%

Si dia uno sguardo al codice **(sono 20 righe!!!)**:

```
install.packages('neuralnet')
library("neuralnet")

#Going to create a neural network to perform sqare rooting
#Type ?neuralnet for more information on the neuralnet library
#Generate 50 random numbers uniformly distributed between 0 and 100
#And store them as a dataframe

traininginput <-  as.data.frame(runif(50, min=0, max=100))
trainingoutput <- sqrt(traininginput)

#Column bind the data into one variable
trainingdata <- cbind(traininginput,trainingoutput)
colnames(trainingdata) <- c("Input","Output")
```

#Train the neural network
#Going to have 10 hidden layers
#Threshold is a numeric value specifying the threshold for the partial
#derivatives of the error function as stopping criteria.

net.sqrt <- neuralnet(Output~Input,trainingdata, hidden=10, threshold=0.01)
print(net.sqrt)

#Plot the neural network
plot(net.sqrt)

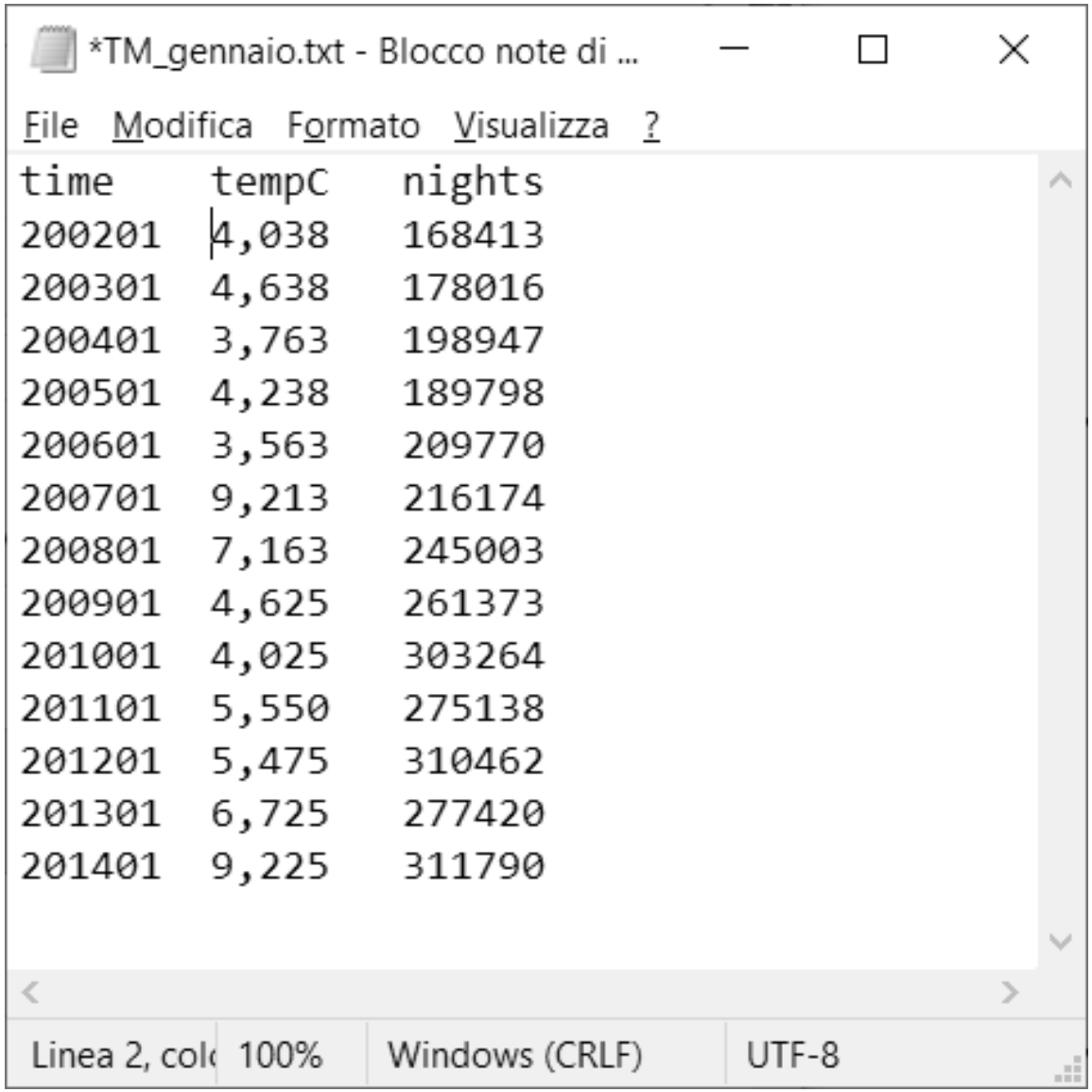
#Test the neural network on some training data
testdata <- as.data.frame((1:10)^2) #Generate some squared numbers
net.results <- compute(net.sqrt, testdata) #Run them through the neural network

#Lets see what properties net.sqrt has
ls(net.results)

#Lets see the results
print(net.results$net.result)

#Lets display a better version of the results
cleanoutput <- cbind(testdata,sqrt(testdata),
as.data.frame(net.results$net.result))
colnames(cleanoutput) <- c("Input","Expected Output","Neural Net Output")
print(cleanoutput)

CREATE UN FILE TXT (da Excel o direttamente in BloccoNote) in cui i valori siano separati da TAB



ALTRO ESEMPIO (come caricare un file) - In rosso le parti opzionali

#la seguente operazione non è più da fare se è già stata eseguita

```
install.packages('neuralnet')
library("neuralnet")
```

Sostituite la parte in giallo con il vostro PATH e usate il backslash invece dello slash

```
#carico un file
dataset<-read.delim2("D:/raff/documents/backup-settembre2012/Documents/Lavoro2014-2018/paper2017/ToSee/TM_gennaio.txt")

#controllo se ci sono dati mancanti:
apply(dataset,2,function(x) sum(is.na(x)))
```

#Normalizzo i dati:

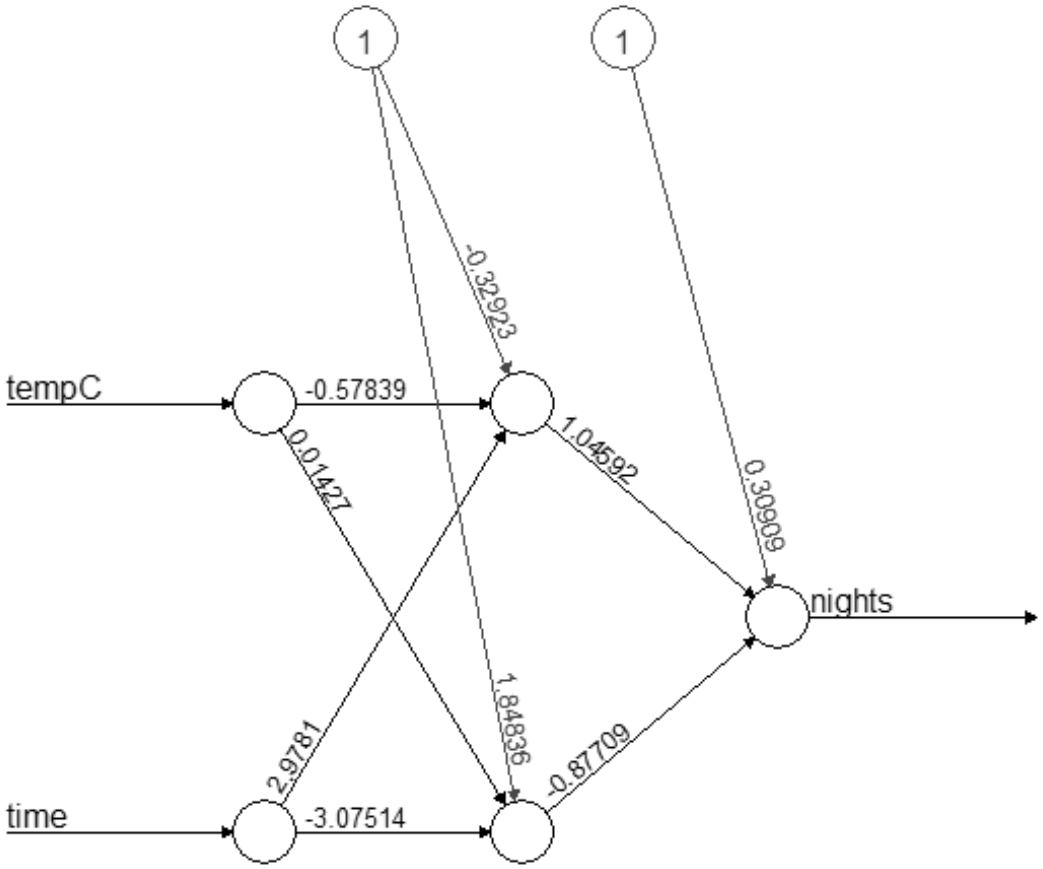
```
index <- sample(1:nrow(dataset),round(0.75*nrow(dataset)))
maxs <- apply(dataset, 2, max)
mins <- apply(dataset, 2, min)
scaled <- as.data.frame(scale(dataset, center = mins, scale = maxs - mins))
train_ <- scaled[index,]
test_ <- scaled[-index,]
```

##creo la rete (errore di 0.01483):

```
library(neuralnet)
n <- names(train_)
nn <- neuralnet(nights ~ tempC + time, data=train_ , hidden=2)
```

#faccio Il plot della rete

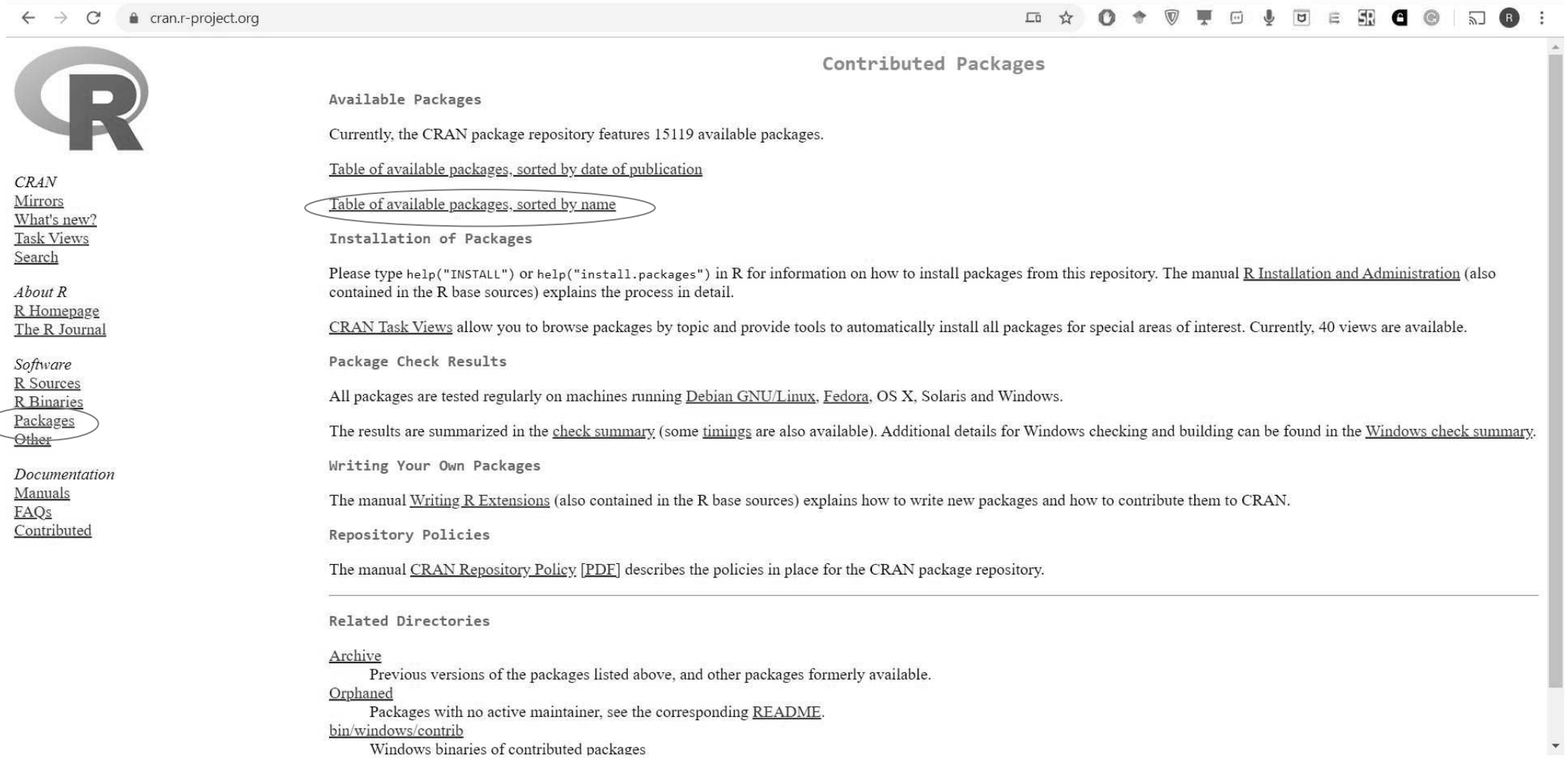
plot(nn)



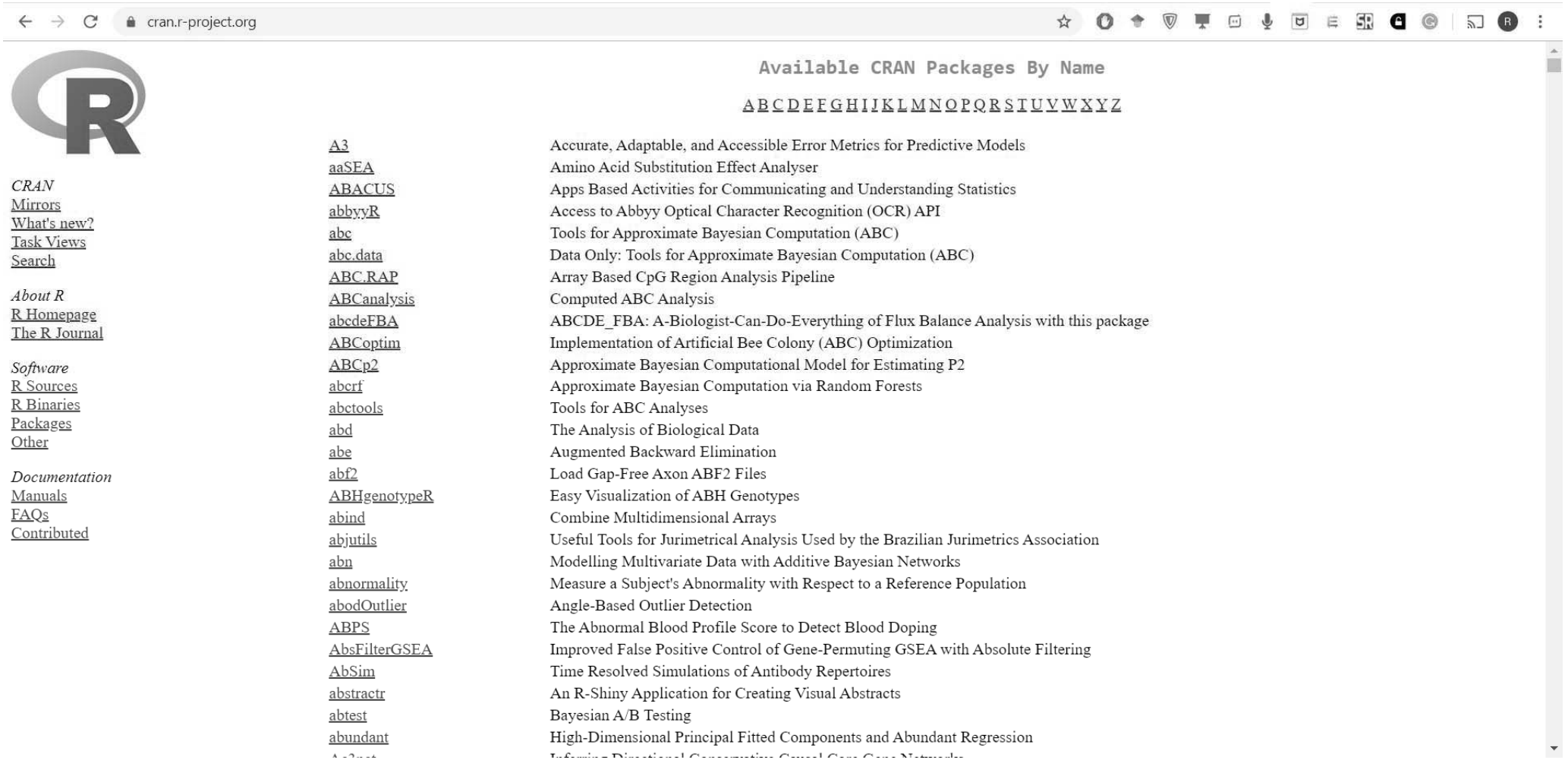
Error: 0.028545 Steps: 45

DESCRIZIONE DEI PACKAGES E DELLE LIBRERIE

Tutti i packages disponibili in R si trovano qui: https://cran.r-project.org/
CLICK SU PACKAGES e, poi, sulla voce packages a sinistra e su Table of available packages, sorted by name



AVRETE L'ELENCO DI TUTTI I PACKAGE DISPONIBILI



Ora con la funzione TROVA del browser, cercate NEURALNET, oppure servitevi degli indicatori alfabetici posti in alto

Noi abbiamo usato NEURALNET, di cui trovate la descrizione qui:

http://cran.mirror.garr.it/mirrors/CRAN/web/packages/neuralnet/neuralnet.pdf

Ma ce ne sono anche altri (es. AMORE, A MORE flexible neural network package)

Provate a dare uno sguardo anche a NeuralNetTools (Visualization and analysis tools to aid in the interpretation of neural network models. Functions are available for plotting, quantifying variable importance, conducting a sensitivity analysis, and obtaining a simple list of model weights).

Descrizione qui:

https://cran.r-project.org/

Risorse su R

Altri tutorial per approfondire:

Engineering Department Cambridge University
http://mi.eng.cam.ac.uk/~mjfg/local/I10/i10_hand4.pdf
Youtube: http://www.youtube.com/watch?v=ZoPJGmpYJzw

L'ambiente R. Buoni manuali:
per principianti: http://rante.org/2011/05/30/un-altro-manuale-per-r/

tutorial:
http://www.ssis.it/modelli/modelli_stat/costruire_modelli_stat_eng/doc/costruire_modelli_stat-44.htm
un altro manuale: http://cran.**r**-project.org/doc/contrib/Mineo-dispensaR.pdf
esempi e applicazioni: http://cran.**r**-project.org/doc/contrib/nozioniR.pdf

pagina principale di R: http://cran.r-project.org/ *(molti package, tra cui ANN)*

altri sw:
http://digilander.libero.it/vvillas/reti_neurali/software_per_reti_neurali.htm

free tool per ANN: http://www.justnn.com/
http://www.bcp.psych.ualberta.ca/~mike/Software/

su Wikipedia altre info:
http://en.wikipedia.org/wiki/Neural_network_software

Classificazione e addestramento: SOM, SVM (possibilità progetto**)**

MACHINE LEARNING

In realtà le ANN vanno ricomprese nel più ampio argomento del Machine Learning.

Il machine learning è una branca dell'Intelligenza artificiale che si occupa della progettazione e dello sviluppo di algoritmi che permettono alle macchine di evolvere le proprie attitudini basandosi su dati empirici, quali quelli provenienti da sensori o da basi di dati.

Un learner (qui si preferirà adottare direttamente il termine inglese) può avvantaggiarsi di esempi (data) per catturare le caratteristiche di interesse di distribuzioni di probabilità sconosciute.

I dati possono essere esempi che illustrano le relazioni tra le variabili osservate.

Un importante obiettivo della ricerca sul machine learning è apprendere automaticamente a riconoscere pattern complessi e prendere decisioni intelligenti basati sui dati; la difficoltà risiede nel fatto che il set di tutte la possibili attitudini dati tutti i possibili input è troppo grande per essere rappresentato dall'insieme degli esempi osservati (training data).

Dunque il learner deve generalizzare dagli esempi dati e deve essere in grado di produrre output utili nei nuovi casi.

Secondo Tom M. Mitchell

“A computer program is said to learn from experience E
with respect to some class of tasks T
and performance measure P,
if its performance at tasks in T, as measured by P,
improves with experience E.”

Generalizzazione

La generalizzazione è l'abilità di un algoritmo di machine learning di funzionare accuratamente su nuovi, mai visti prima, esempi dopo il training su un data set finito.

L'obiettivo principale di un learner è generalizzare dalla sua esperienza.

Gli esempi di training provengono da alcune non conosciute probabilità di distribuzione e il learner deve estrarre da questi qualcosa di più generale, qualcosa circa la distribuzione, che gli permetta di produrre risposte utili in casi nuovi.

Machine learning e data mining

Generalmente c'é confusione tra questi termini poiché spesso adottano gli stessi metodi e vanno in sovrapposizione. Ad ogni modo, per distinguerli, basta considerare che:

- Il machine learning ha il focus sulla predizione, basata su proprietà conosciute apprese dai dati di training
- Il data mining (che è la fase di analisi del knowledge discovery in un database) ha il focus sulla scoperta di (preventivamente) sconosciute proprietà dei dati

Ad ogni modo, le due aree hanno parecchie sovrapposizioni: il data mining usa molti metodi di machine learning, ma spesso con differenti obiettivi.

Interazione dell’uomo

Alcuni sistemi di machine learning tendono ad eliminare il bisogno dell’intuizione umana nell’analisi dei dati, mentre altri adottano un approccio collaborativo tra uomo e macchina.

L’intuizione umana non può, comunque, essere interamente eliminata, poiché i progettisti dei sistemi devono comunque specificare come i dati devono essere rappresentati e quali meccanismi saranno usati per ricercare la caratterizzazione dei dati.

Tipi di algoritmi

Gli algoritmi di machine learning possono essere organizzati in una tassonomia basata sul risultato desiderato:

- Apprendimento supervisionato (supervised learning): genera una funzione che “mappa” gli input agli output desiderati (chiamati anche etichette, perché spesso sono definite da esperti umani che etichettano gli esempi di training. Ad esempio in un problema di classificazione, il learner approssima una funzione che mappa un vettore in classi secondo alcuni esempi di input-output della funzione.
- Apprendimento non supervisionato (unsupervised learning): modella un insieme di input, quali il clustering.
- Apprendimento semi-supervisionato (semi-supervised learning): combina sia esempi etichettati che non etichettati per generare una appropriata funzione o classificatore

- Apprendimento con rinforzo (reinforcement learning): apprende come agire data un'osservazione del mondo, Ogni azione ha qualche impatto sull'ambiente, e l'ambiente restituisce dei feedback che guidano l'algoritmo di apprendimento.
- Inferenza trasduttiva (Transduction o trasductive inference): cerca di predire nuovi output sucasi specifici e fissati (test) da casi osservati e specifici (training)
- Imparare ad apprendere (learning to learn): apprende il suo bias induttivo sulla base di precedenti esperienze

Teoria

L’analisi computazionale degli algoritmi di machine learning e la loro performance è una branca della theoretical computer science conosciuta come computational learning theory.

Dato che i training set sono finiti e il futuro è incerto, la learning theory di solito non garantisce la performance degli algoritmi.

Invece, i confini probabilistici delle performance sono piuttosto comuni.

In aggiunta ai confini della performance, i teorici del computational learning studiano la complessità temporale e la fattibilità di apprendimento.

Nella teoria del computational learning, una computazione è considerata fattibile se può essere compiuta in un tempo polinomiale.

Vi sono due tipi di risultati di complessità temporale:
1) i risultati positivi mostrano che una certa classe di funzioni può essere appresa in un tempo polinomiale;
2) i risultati negativi mostrano che certe classi on possono essere apprese in un tempo polinomiale.

Vi sono molte similarità tra la teoria del machine learning e la statistica, così spesso i due termini sono utilizzati quasi come sinonimi.

Approcci del machine learning

Decision tree. Viene utilizzato un decision tree come modello predittivo che "mappa" le osservazioni circa un'istanza con le conclusioni sul valore target dell'istanza stessa.

Association rule. E' un metodo per scoprire relazioni tra variabili in database di grandi dimensioni.

Artificial neural network. Le computazioni sono strutturate in termini di gruppi interconnessi di neuroni artificiali, che processano informazioni utilizzando un approccio connessionista alla computazione. Le reti neurali moderne sono strumenti di modellazione dei dati statistiche, non lineari. Utilizzate per modellare relazioni complesse tra input e output, per cercare pattern nei dati, o per catturare la struttura statistica di un distribuzione di probabilità congiunta tra variabili osservate.

Programmazione genetica. E' una metologia basata su algoritmi evolutivi ispirati dall'evoluzione biologica per trovare programmi che compiano un task definito.
E' una tecnica di machine learning utilizzata per ottimizzare una popolazione di programmi in accordo con uno scenario determinato dall'abilità del programma di compiere un determinato task computazionale.

Programmazione logica induttiva (inductive logic programming). E' un approccio alle regole di apprendimento utilizzando la programmazione logica come rappresentazione uniforme degli esempi, della conoscenza di background e delle ipotesi. Data una codifica della conoscenza di base ed un set di esempi rappresentati come un database logico di fatti, il sistema deriva un programma logico di ipotesi che si addestra con tutti gli esempi positivi e nessuno dei negativi.

Support Vector Machine (SVM).
E’ un insieme di metodi di apprendimento supervisionato (ma oggi ve ne sono anche esempi non supervisionati) utilizzati per classificazione e regressione.

Dato un set di esempi di training, ognuno marcato come appartenente ad una delle due (o più) categorie, un algoritmo di training dell’SVM costruisce un modello che predice se uno dei nuovi esempi cade in una categoria o nell’altra.

Clustering. La cluster analysis è l'assegnazione di un insieme di osservazioni in sottoinsiemi (chiamati cluster) composti in modo che le osservazioni in uno stesso cluster sono simili in qualche maniera, mentre le osservazioni in cluster differenti sono dissimili.
La varietà di tecniche di clustering dipende da diffferenti assunzioni sulla struttura dei dati, spesso definita da qualche metrica di similitudine e valutata per esepio dalla internal compactness (similitudine tra membri dello stesso cluster) e separation tra cluster differenti.
Altri metodi sono bassati sulla estimated density e sulla graph connectivity.
Il clustering è un metodo di apprendimento non supervisionato e una comune tecnica per l'analisi statistica dei dati.

Reti Bayesiane (Bayesian networks). E' un modello grafico probabilistico che rappresenta un insieme di variabili random e le loro indipendenze condizionali tramite un grafo aciclico diretto (DAG).

Per esempio, una rete bayesiana può rappresentare la relazione probabilistica tra malattia e sintomi.

Dati i sintomi, la rete può essere usata per calcolare la probabilità della presenza di varie malattie.

Esistono algoritmi efficienti che compiono inferenza e apprendimento.

Apprendimento con rinforzo (reinforcement learning). Tratta come un agente deve intraprendere azioni in un ambiente per massimizzare alcune nozioni di ricompensa a lungo termine.

Gli algoritmi di apprendimento con rinforzo cercano di trovare una policy che “mappi” gli stati del mondo sulle azioni che gli agenti devono intraprendere in tali stati.

Differisce dall’apprendimento supervisionato perché le coppie input/output corretti non sono mai presentati né azioni subottimali vengono esplicitamente corrette.

Apprendimento per rappresentazione (representation learning). Molti algoritmi di apprendimento, per la maggior parte di apprendimento non supervisionato, tendono a scoprire le migliori rappresentazioni degli input forniti durante il training. Esempi classici includono la principal component analysis e la cluster analusis. Gli algoritmi di apprendimento per rappresentazione spesso cercano di preservare le informazioni negli input ma le trasformano in un modo che le rende utili, spesso durante una fase di pre-processing prima di operare la classificazione o la predizione, permettendo di ricostruire gli input provenienti dai dati generati da una distribuzione sconosciuta. Vi sono tre principali tipi di algoritmi: i manifold learning, gli spars coding, i deep learning algorithms (approfondire, se interessa).

Sparse Dictionary Learning.
Nell’apprendimento Sparse Dictionary un dato è rappresentato come una combinazione lineare di funzioni di base ed i coefficienti sono assunti sparsi.
(approfondire, se interessa)

Applicazioni

Attualmente il machine learning viene applicato a molti campi:

machine perception
natural language processing syntactic
search engines
bioinformatics
cheminformatics
stock market analysis
Sequence mining
game playing
object recognition in computer vision
robot locomotion
structural health monitoring
Sentiment Analysis (Opinion Mining)
computer vision
pattern recognition
medical diagnosis
brain-machine interfaces
Detecting credit card fraud
Classifying DNA sequences
speech and handwriting recognition
software engineering
adaptive websites
computational finance
affective computing
Recommender systems

Software

Vi sono vari tipi di software che contengono un’ampia varietà di metodi di machine learning.

Tra questi ne ricordiamo alcuni: RapidMiner, LIONsolver, KNIME, Weka, ODM, Shogun toolbox, Orange, Apache Mahout, scikit-learn, mlpy.

Chi era John Henry Holland? Approfondire
Chi era David B. Fogel? Approfondire